Janne Kauhanen

Opetetun objektin tunnistus tietokone-näöllä käyttäen Haar-koulutusta

Helsinki Metropolia Ammattikorkeakoulu

Tietotekniikka

Insinöörityö

8. marraskuuta 2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Janne Kauhanen Opetetun objektin tunnistus tietokonenäöllä käyttäen Haar-koulutusta 48 sivua + 1 liite 8.11.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Timo-Pekka Jäntti Kielenohjaaja Jussi Alhorinne

Tämän opinnäytetyön tavoitteena on parantaa omaa osaamistani tietokonenäköä käyttävissä sovelluksissa ja selvittää, miten tehokkaasti ja tarkasti ohjelmalle koulutetun objektin tunnistusta voidaan käyttää käytännön sovelluksissa. Työssä tehtiin viisi testikoulutusta kahdella eri kategorialla: yhdessä kaskaadille opetetaan Haar-koulutuksella yksinkertaisia 2d-kuvioita, jotka pitäisi voida kameralta tunnistaa, ja toisessa Haar-koulutuksella on ohjelmalle opetettu monimutkaisempi 3d-objekti, joka pitää tunnistaa erilaisissa tilanteissa.

Käytän työssä OpenCV-kirjastoa, joka on ohjelmoitu C- ja C++-ohjelmointikielillä, mutta jolla on rajapintoja muihinkin ohjelmointikieliin. Pyrin käymään nopeasti läpi, miten tätä kirjastoa käytetään ja mitä sen osia Haar-koulutus käyttää, mutten syvällisemmin sen toimintaan paneudu. Työn alussa käyn läpi, miten tietokonenäkö käytännössä toimii ja mitä tekniikoita siihen liittyy, jotta myöhemmät osuudet olisivat helpommin ymmärrettävissä.

Työn lopputuloksena syntyi koulutuskaskaadi, joka pystyy tunnistamaan opetettuja objekteja useimmista tilanteista, kunhan kamera on samaa laatua kuin koulutuksessa käytetty.

Avainsanat	tietokonenäkö, OpenCV, Haar, objektin koulutus, objektin tunnistus

Author(s) Title Number of Pages Date	Janne Kauhanen Detection of a taught object with computer vision by using Haar-training 48 pages + 1 appendix 8 November 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Timo-Pekka Jäntti, Lecturer Jussi Alhorinne, Language instructor

The aim of this Bachelor's thesis is to improve my own skills in applications that utilize computer vision, and find out how effectively and accurately taught object identification can be used in practical applications. Five example trainings were done during the thesis, in two different categories: In one I taught simple 2d-shapes via Haar-training that the program should identify from a videofeed, and the other was taught more complex 3d-objects via Haar-training that the program should identify in all situations.

I will be using the OpenCV-library in the thesis, that has been programmed using C- and C++ programming languages, but with additional interfaces for other languages. I will attempt to quickly go through how this library is used, and what parts of it Haar-training utilizes, but I will not go in-depth into it. In the beginning of the thesis I will go through how computer vision works in practice, and what techniques it involves to enable the subsequent parts of the thesis to be easier to understand.

The final result is a training cascade that is able to recognize taught objects from most situations, as long as the camera quality is the same that was used for the training.

Keywords	computer vision, OpenCV, Haar-training, training an object, object detection

Sisällysluettelo

Liitteet

Liite 1. Käytettyjä negatiivisia kuvia

1 Johdanto

Olen tutkinut ja ollut kiinnostunut tietokonenäöstä jo jonkin aikaa, siitä lähtien kun opiskelin sitä puoli vuotta opiskelijavaihdon aikana Japanissa. Aihe on vieläkin niin kiinnostava ja laaja, että saisin siitä mielenkiintoisen insinöörityön aiheen. Ohjelman kouluttaminen tietokonenäöllä tunnistamaan tiettyjä objekteja annettavasta syötteestä, kuvista tai suorasta videokuvasta, on miellyttävä aihe, joka on mahdollista toteuttaa parillakin eri metodilla.

Haluan tällä työllä selvittää, miten helppoa on erilaisista kuvatilanteista, kulmista ja valaistuksesta jonkin tietyn ohjelmalle opetetun esineen tunnistaminen. Tämä työ, ja siihen liittyvä aihealueen tutkiminen, toivottavasti antaa minulle uutta tietoa tietokonenäön sovelluksista, ja varsinkin siitä, miten laajalti opetetun objektin tunnistusta voi kehittää. Pitkän tähtäimen tavoitteena olisikin kehittää koulutusohjelma, jolla voitaisiin tunnistaa monia erilaisia objekteja kuvasta/videolta ja antaa näistä palautetta ohjelman käyttäjälle. Tähän tavoitteeseen en kuitenkaan tässä insinöörityössä suoranaisesti pyri vaan selvitän perusteet tähän tavoitteeseen pääsemiseksi. Käyn myös läpi käyttämäni ohjelmakirjaston, OpenCV:n, pääasiat sekä yleisiä asioita siitä, miten tietokonenäkö toimii. Näitten perusasioiden lisäksi käyn läpi käyttämääni koulutusalgoritmia, eli kuinka Haar-koulutus toimii ja miten sillä tehdään kaskaaditiedosto, jota OpenCV voi suoraan käyttää tunnistamaan haluttua objektia.

Tällä työllä haluan siis vastauksen kysymyksiin: Kuinka nopea ja tarkka on koulutetun objektin tunnistus suorasta videokuvasta? Miten optimaalinen tämä koulutusmetodi on, ja soveltuuko se käytännön sovelluksiin? Miten Haar-koulutus itse asiassa toimii?

2 Tietokonenäkö

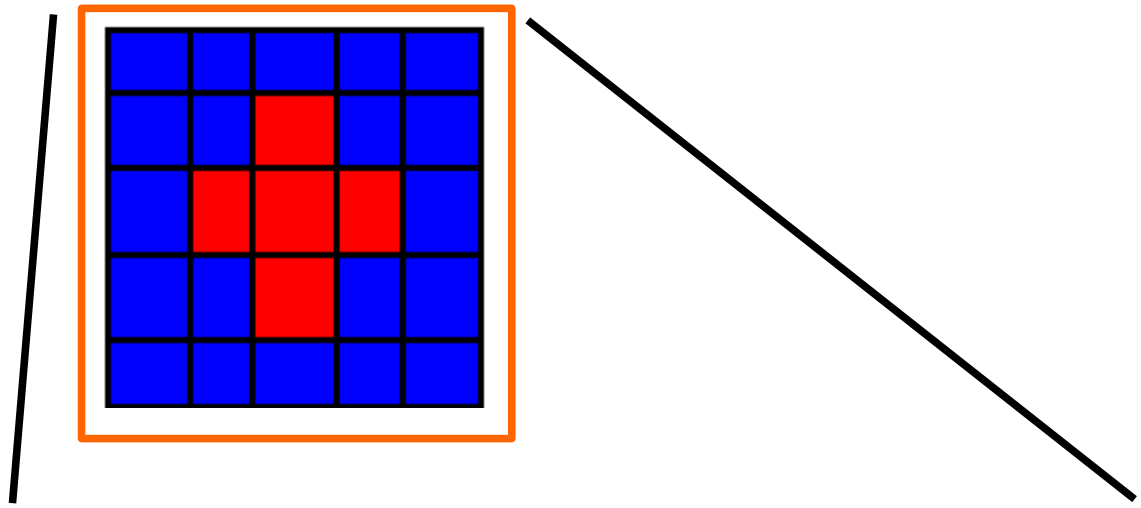
Tietokonenäöstä puhuttaessa viitataan yleensä kuvasta tai videokamerasta saadun datan muuttamista päätökseksi tai sen edustamiseksi uudella tavalla [1, s. 2]. Saadusta datasta syntyvä päätös voi olla esim. "Tässä kuvassa on ympyrä" tai "Tämä kuva on vihreä", kun taas datan uudella tavalla edustaminen voi olla vaikkapa värikuvan muuttamista harmaasävyiseksi. Tietokonenäöllä pyritään ohjelmallisesti toistamaan sitä, miten me omilla silmilllä näkisimme jonkin tietyn asian. Koska useimmille ihmisille näkeminen on itsestäänselvä asia, on sitä vaikea tunnistaa erittäin hankalaksi asiaksi. Meille on kehittynyt evoluution saatossa erittäin monimutkainen prosessointijärjestelmä aivoihimme, jonka avulla voimme tunnistaa värit, muodot ja valon vaihtelun pelkällä silmäyksellä. Tietokonenäössä kuitenkin ohjelma saa kuvasta tai kameralta vain taulukon numeroita, joiden merkitys joudutaan ohjelmallisesti selvittämään.

Tietokonenäköä käytetään nykyään erittäin monilla kentillä, kännyköiden kasvontunnistuslukoista erilaisten syöpäsolujen skannaukseen lääketutkimuksessa. Koska tietokonenäköä soveltavien teknologioiden käyttö on erittäin laajalle levinnyttä ja niiden käyttöaste jatkaa vieläkin kasvua, ovat myös erilaiset ohjelmalliset tietokonenäköalgoritmit kehittyneet paljon. Mitä enemmän tämän teknologian käyttö kasvaa, sitä parempia algoritmejä me sen käyttöön tarvitsemme.

Työkalu, jota pääasiassa käytän tässä insinöörityössä, on OpenCV-ohjelmointikirjasto, joka on vapaan lähdekoodin kirjasto juuri tietokonenäön sovelluksiin. [1, s. 1.]

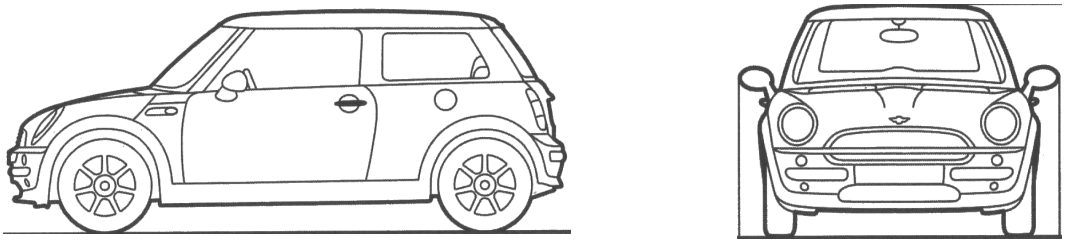
Tietokonenäkö saa saamistaan kuvista käyttöön vain taulukon kuvan pikseleiden arvoista, kuten kuvasta 1 voimme nähdä, ja tämän datan perusteella rakennetaan käyttäjille näytettävä kuva. Jokainen pikseli pitää sisällään 3 arvoa: Sininen, vihreä ja punainen. Yleisemmästä RGB (red, green, blue) -väriskeemasta poiketen OpenCV käyttää BGR:ää (blue, green, red). Näitä numeroarvoja muuttamalla voidaan kuvaa muokata eri muotoihin, ja niistä voi algoritmeilla päätellä, onko kuvassa yhtenäisiä viivoja tai muotoja. Nämä algoritmit ovat yleiseltä nimeltään hahmontunnistusalgoritmeja, ja niitä voidaan soveltaa monilla aloilla. [1, s. 3.]

Hahmontunnistuksessa on kuitenkin monia ongelmia, esimerkiksi miten tunnistaa sama objekti/objektit, kun perspektiivi vaihtuu, kuten kuvassa 2 tapahtuu? Tätä voidaan lievittää opettamalla sama objekti ohjelmalle monesta kulmasta, mutta se on silti sinnikäs ongelma.



255	0	0	255	0	0	255	0	0	255	0	0	255	0	0
255	0	0	255	0	0	0	0	255	255	0	0	255	0	0
255	0	0	0	0	255	0	0	255	0	0	255	255	0	0
255	0	0	255	0	0	0	0	255	255	0	0	255	0	0
255	0	0	255	0	0	255	0	0	255	0	0	255	0	0

Kuva 1. Esimerkki kuinka tietokonenäkö "näkee" pikselit yksittäisestä kuvasta.



Kuva 2. Vasemmanpuoleinen kuva on opetettu, ja ohjelma tunnistaa sen. Oikeanpuoleinen kuva ei tunnistu ollenkaan, paitsi jos sekin on opetettu.

OpenCV lyhyesti

Olin tätä työtä ennen käyttänyt jo OpenCV-kirjastoa ollessani opiskelijavaihdossa, joten se oli jo valmiiksi lyhyellä listalla työkaluja, joita ajattelin käyttää tässä työssä. Pienen budjetin takia en voinut maksullisia työkaluja edes harkita, joten muut työkalut, joita harkitsin tähän työhön, olivat pakostakin muita vapaan lähdekoodin työkaluja. OpenCV:n lisäksi harkitsin aluksi Integrating Visual Toolkit (IVT) -työkalua, joka on myös va-

paan lähdekoodin ohjelma. Tämä olisi kuitenkin vaatinut monen OpenCV:stä eroavan asian uudelleenopettelua, joten päädyin työn toteuttamiseen OpenCV:llä. Toinen vaihtoehto olisi ollut SimpleCV, mutta se on toteutettu Pythonilla, johon itselläni ei ole erityisesti kosketusta. Jos Pythonia osaisin, SimpleCV:llä saattaisi olla helpompi toteuttaa sovelluksia, varsinkin koska se pitää sisällään yhteydet OpenCV:n kirjastoihin.

Kuten mainittua aikaisemmin, OpenCV on vapaan lähdekoodin C/C++ -ohjelmointikirjasto juuri tietokonenäkösovellusten käyttöön, ja toimii Windowsilla, Linuxilla, OSX:llä sekä Androidilla. Sille löytyy C/C++:n lisäksi rajapintoja myös Pythoniin, Rubyyn, Matlabiin ynnä muihin ohjelmointikieliin, joiden kehitys on vieläkin etenemässä. OpenCV kehitettiin tarjoamaan yhteinen infrastruktuuri tietokonenäön sovelluksiin ja vauhdittamaan niitten käyttöä kaupallisissa tuotteissa. [1, s. 1; 4.]

OpenCV sisältää yli 2500 optimoitua tietokonenäön algoritmia, joihin sisältyy normaaleja ja tarkempia tietokonenäön ja koneoppimisen algoritmeja. Myös Haar-koulutus sisältyy jo valmiiksi OpenCV:n kirjastoihin. Näitä algoritmeja voidaan käyttää kasvojen ja objektien tunnistukseen, ihmisten liikkeiden seurantaan, kameran liikkumisen seurantaan, 3d-mallien luontiin kuvasta, punaisen värin poistoon silmistä valokuvan salaman jäljiltä yms. OpenCV on isolta osin sen käyttäjien luoma ja laajentama työkalu, ja sillä onkin yli 47 tuhatta käyttäjää sen omassa yhteisössä. Kirjastoa käytetään myös laajalti yritysten, tutkijoiden ja valtioiden keskuudessa. OpenCV kehitettiin alunperin Intelin toimesta, ja on sittemmin robotiikan tutkimusyrityksen Willow Garagen sekä tietokonenäköön keskittyneen yrityksen Itseez:in tukema. [4.]

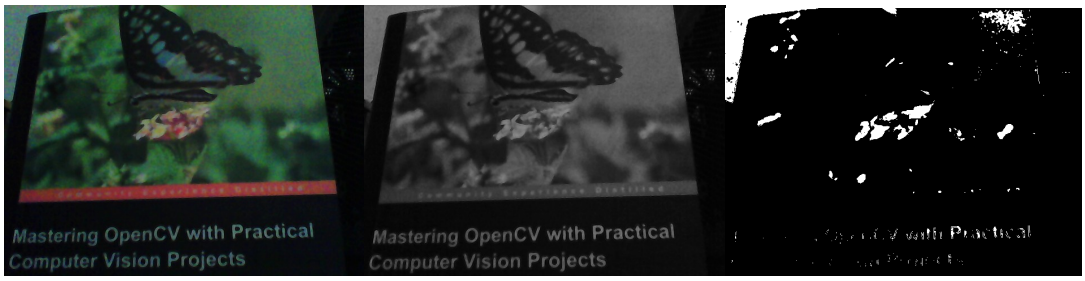
OpenCV:n uusin versio on 3.0 alpha, mutta sen ollen alpha-luontoinen käytän vanhempaa versiota 2.4.9 [5]. Pääosan työn vaatimasta koodauksesta teen Visual Studio 2012, ja joitakin pienempiä osia Visual Studio 2010 Expressillä. Käytän työssä sekä Windows 8.1 64-bit- että Windows 7 64- versioita.

3 Tietokonenäön perusasioita

Käyn tässä luvussä läpi tietokonenäköön liittyviä perusasioita, joita työssäni käytän, jotta lukija ymmärtäisi niitä myöhemmin käytettäessä. Nämä tekniikat ovat yleisessä käytössä tietokonenäön sovelluksissa.

3.1 Binäärinäkö

Kun puhutaan binäärisestä kuvasta, tarkoitetaan kuvaa, jossa aiemmin puhutussa tietokoneen näkemässä taulukossa on olemassa vain kaksi arvoa per pikseli. Nämä arvot ovat yleensä musta (0) ja valkoinen (1), mutta muitakin arvoja voidaan käyttää.



Kuva 3. Esimerkki binäärikuvasta värikuvan ja harmaasävyisen kera.

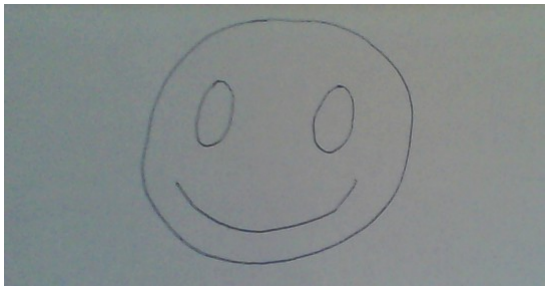
Pääasia on, että vain kahta arvoa käytetään kuvaamaan yhden pikselin väriä. Binäärikuva muodostetaan väri- tai harmaasävykuvasta asettamalla niille erilaiset kynnysarvot (thresholding) eri arvojen erottamiseksi. [10.]

Kuvan 3 esimerkki on toteutettu suorasta videokuvasta, ja voimme tästä kuvasta myös nähdä, kuinka paljon hyvällä valaistuksella on merkitystä kuvan muotojen havaitsemiseen kynnysarvojen määrityksen jälkeisessä binäärikuvassa. Myös kameran laatu on erittäin tärkeä varsinkin opetuksessa, koska jos objekti on opetettu eri kameralla kuin millä sitä yritetään havaita, tulee vastaan havainto-ongelmia.

3.2 Kynnysarvojen asettaminen (thresholding)

Kynnysarvojen asetus (thresholding) on yksinkertaisin kuvan segmentoimistekniikka eli tapa, jolla digitaalinen kuva jaetaan eri pikselijoukkoihin. Kynnysarvojen asetuksella muodostetaan kuvasta, yleensä harmaasävykuvasta, binäärikuva [7]. Kynnysarvojen asetus on erittäin käyttökelpoinen tekniikka esimerkiksi tilanteessa, jossa halutaan erottaa tumma objekti vaaleasta taustasta. Asettamalla oikeat kynnysarvot voidaan tämä tumma objekti erottaa aina taustasta riippumatta siitä, mitä tummaa väriä objektissa on, kunhan näiden värien arvot ovat korkeampia kuin kynnysarvo. Esimerkin mukaisessa tapauksessa siis skannattaisiin jokin tietty kuva RGB-värillisenä, muutettaisiin se harmaasävyiseksi koneella, ja asetettaisiin kynnysarvot siten, että kaikki arvot, jotka ovat alle 20, hylätään. Tällä kynnysarvolla saamme ison skaalan vaaleita värejä hylättyä yhdellä kertaa. Kynnysarvot asettamalla siis voidaan kategorisesti hyväksyä jotkin pikselit kynnysarvon yläpuolella, samalla kun sen alapuolelle jääneet pikselit

hylätään. [1, s. 135.] Hyväksymällä ja hylkäämällä kuvan pikseleitä voimme suoraan käsitellä näitä pikseleitä kahtena suurena joukkona. Muita käyttökohteita tälle tekniikalle on esimerkiksi kirjainten automaattinen tunnistus paperilta, sähkökaavioiden tarkastaminen, karttojen prosessointi ym.



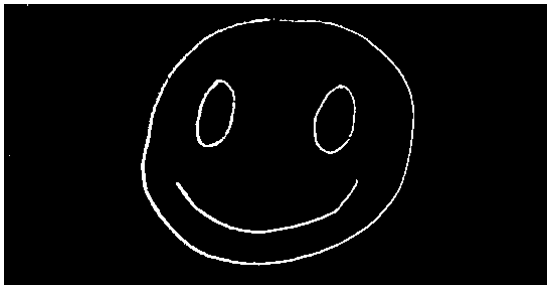
Kuva 4. Esimerkkikuva jota käytettiin kynnysarvojen asettamiseen.

OpenCV tarjoaa omasta puolestaan käyttöömme suoraan funktion *cvthreshold,* joka on uudemmissa versioissa pelkkä *threshold,* jolla voidaan käyttää seuraavia viittä erilaista kynnysarvojen asetusmetodia [7]. Esimerkkikuvana on käytetty kuvaa 4, joka kaapattiin suorasta kamerakuvasta ja jonka kokoa muokattiin irfanView-kuvankäsittelyohjelmalla, kynnysarvona 120 ja maksimikynnysarvona on 255.

Threshold Binary

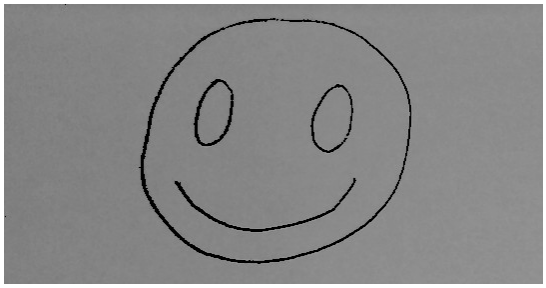
Jos jonkin tietyn pikselin intensiteetti, vahvuus, on korkeampi kuin annettu kynnysarvo, uuden pikselin intensiteetiksi asetetaan maksimiarvo. Muuten pikseli saa arvon nolla. Eli jos kynnysarvo on 30 ja pikselin intensiteetti on 80, se saa arvon 255, jos se on asetettu maksimiarvoksi.

Threshold Binary, Inverted



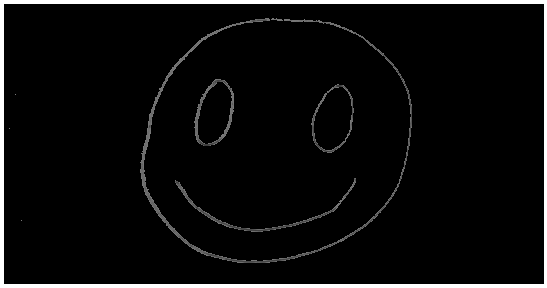
Sama kuin *Threshold Binary*, mutta käänteisesti: Jos pikselin intensiteetti on korkeampi kuin kynnysarvo, se saa arvon nolla. Muuten pikseli saa maksimiarvon.

Threshold to Zero



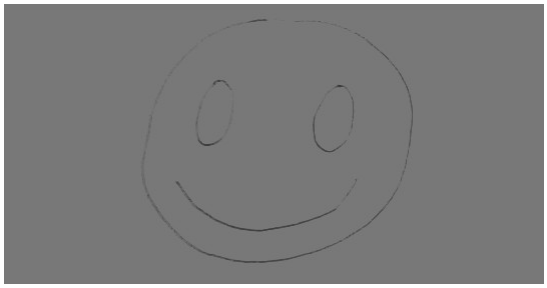
Jos pikselin intensiteetti on alle kynnysarvon, asetetaan pikselin intensiteetti nollaksi. Kynnysarvon ylittävät arvot eivät muutu.

Threshold to Zero, Inverted



Sama kuin *Threshold to Zero*, mutta käänteisesti: Kaikki pikselit, joiden intensiteetti ylittää kynnysarvon, saavat arvon nolla. Kynnysarvon alittavat arvot eivät muutu.

Truncate



Jos pikselin intensiteetti menee yli kynnysarvon, asetetaan pikselin intensiteetti kynnysarvoksi. Jos se jää alle kynnysarvon, pikselin intensiteetti ei muutu.

Esimerkkikuva erilaisilla kynnysarvoilla

Esittelen tässä kuvasta 4 eri kynnysarvojen asetusmetodeilla saatavat kuvat eri kynnysarvoilla taulukkomuodossa niiden erojen havainnoinnin helpottamiseksi. Taulukossa 1 käytetään kynnys-arvomaksimia 255, ja kynnysarvoja 0, 60, 120, 190 ja 255.

Taulukko 1. Kynnysarvojen asetus maksimikynnysarvolla 255.

	0	60	120	190	255
Binary					
Binary Inverted					
Threshold To Zero					
Threshold To Zero Inverted					
Truncate					

Taulokosta 1 näemme siis nopeasti, millaista jälkeä erilaiset kynnysarvojen asetusmetodit tuottavat. Työn loppuosassa käytämme eniten *Binary*- ja *Binary Inverted*-metodeja. *Binary*-metodissa voimme nähdä, että kun kynnysarvona on 0 ja maksimiarvona 255, muutetaan kuvan jokainen pikseli valkoiseksi (arvo 255), koska kaikkien pikseleiden arvot ovat yli tai sama kuin kynnysarvo, kun taas *Binary Inverted* muuttaa ne käänteisesti mustiksi (arvo 0). Kun taas nostamme kynnysarvoa 60:een, voimme nähdä kummillakin metodeilla jo muodostuvan pikseleistä kuvaa, koska näiden pikseleiden arvot jäävät kynnysarvon alapuolelle, eli ne ovat tummempia kuin muut kohdat. Jos kuvassa olisi parempi valaistus, voisimme käyttää tarkempia arvoja kuin tässä on käsitelty mutta normaaleissa tilanteissa ei usein ole optimaalinen valaistus, joten laajemman skaalan läpikäyvät esimerkit ovat hyödyllisempiä. Taulukossa 2 näemme, mitä tapahtuu samoilla metodeilla, jos puolitamme maksimikynnysarvon 127:ään.

Taulukko 2. Kynnysarvojen asetus maksimikynnysarvolla 127.

Koska tämän sarjan maksimiarvo on määrätty 127:ään, ei täysin valkoisia kuvia esiinny. Esimerkiksi *Binary*-metodilla toteutetussa sarjassa on 0 - 120 kynnysarvoisissa ku-

vissa valkoinen taustaa ei koskaan muuteta arvoon 255, koska maksimiarvo on vain 127. Jos kuvien tausta olisi jo alkuperäisessä kuvassa puhtaan valkoinen, ei se kuitenkaan siitä muuttuisi, jos kynnysarvo ei ylity.

3.3 Morfologia, eroosio ja dilaatio

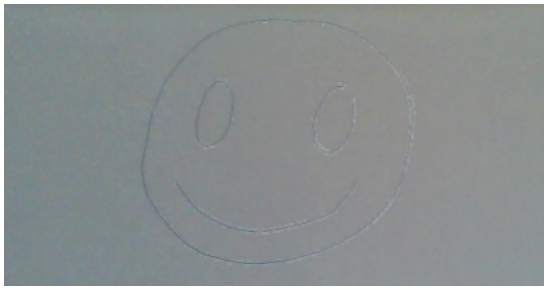
Dilaatio ja eroosio ovat tietokonenäössä kaksi erittäin yleistä morfologian operaattoria. Ne ovat kummatkin niin sanottuja *morfologisia filttereitä* (*morphological filters*). Kun puhutaan morfologiasta tietokonenäön yhteydessä, tarkoitetaan matemaattista morfologiaa, jota käytetään geometristen rakennelmien analyysiin ja prosessointiin. Tätä tekniikkaa käytetään useimmiten binäärikuvien analyysiin ja prosessointiin, mutta voidaan käyttää myös harmaasävykuviin. Tärkein elementti matemaattisessa morfologiassa on *jäsentävä elementti* (*structuring element*), joka on yksinkertainen ennalta määrätty muoto. Se tunnetaan myös nimellä *kernel,* ja sen tarkoituksena on vetää johtopäätöksiä siitä, kuinka se sopii tai ei sovi kuvassa oleviin muotoihin. Kernelillä on *ankkuripiste* (anchor point) sen keskikohdassa, joka määrää mihin kohtaan muoto kuvassa sijoittuu. Kernelin muotona käytetään usein jotain erittäin yksinkertaista, kuten neliötä tai ympyrää. [9.]

OpenCV sisältää valmiiksi rakennetut funktiot eroosiolle ja dilaatiolle, ja ne ovar *erode* ja *dilate.* Bradskin ja Kaehlerin (2008) kuvaus näille operaatioille on seuraava:

> Dilaatio on jonkin kuvan (tai kuvan alueen) muokkausta jollakin *kernelillä.* Tämä kernel voi olla minkä tahansa muotoinen tai kokoinen, kunhan sillä on yksi määrätty *ankkuripiste* (anchor point). Useimmiten kernel on pieni kokonainen neliö tai ympyrä, jonka ankkuripiste on sen keskipisteessä. Kernelin voidaan ajatella olevan malli tai maski ja se vaikuttaa dilaatioon *paikallisen maksimin* (local maximum) operaattorina. Kun kernel skannataan kuvan päälle, laskemme kernelin kattavan alueen pikselien maksimiarvot ja korvaamme ankkuripisteen alla olevan pikselin arvon tällä maksimiarvolla. Tämä kasvattaa kuvan kirkkaiden alueiden kokoa. Dilaatio-operaation nimi juontaa tästä kasvusta. Eroosio on päinvastainen operaatio, ja siinä lasketaan kernelin alueelle *paikallista minimiä* (local minimum). Eroosio luo uuden kuvan alkuperäisestä käyttäen seuraavaa algoritmia: kun kernel skannataan kuvan päälle, laskemme sen alueelta pienimmän mahdollisen pikseliarvon ja korvaamme ankkuripisteen alla olevan pikselin arvon tällä arvolla. Yleisesti ottaen dilaatio kasvattaa aluetta, ja eroosio pienentää aluetta. Dilaatio myös silentää koveruuksia ja eroosio silentää pullistumia. Se mitä dilaatio ja eroosio

> kuitenkin tarkalleen tekevät riippuu kernelistä, mutta mainitut asiat ovat yleisesti tosia useimmissa kerneleissä. [1, s. 115 - 116.]

Dilaatio siis tarkoittaa alueen kasvatusta, eli jos tausta on valkoinen, kasvaa tämä valkoinen alue pienentäen tummia alueita. Eroosio taas tekee tämän toiseen suuntaan, eli kasvattaa tummia alueita. Eroosiolla voimme siis vahventaa pieniä piirteitä kuvasta tuoden ne paljon näkyvämmin esiin. Jos taas haluamme putsata kuvaa poistaen pieniä alueita, likaa tai vastaavaa, tämän voimme tehdä dilaatiolla. Alla olevissa kuvissa 5 ja 6 olen tehnyt kuvasta 4 eroosion ja dilaation, neliön muotoisella 2 x 2 kernelillä.



Kuva 5. Dilaation jälkeinen kuva.

Kuva 6. Eroosion jälkeinen kuva.

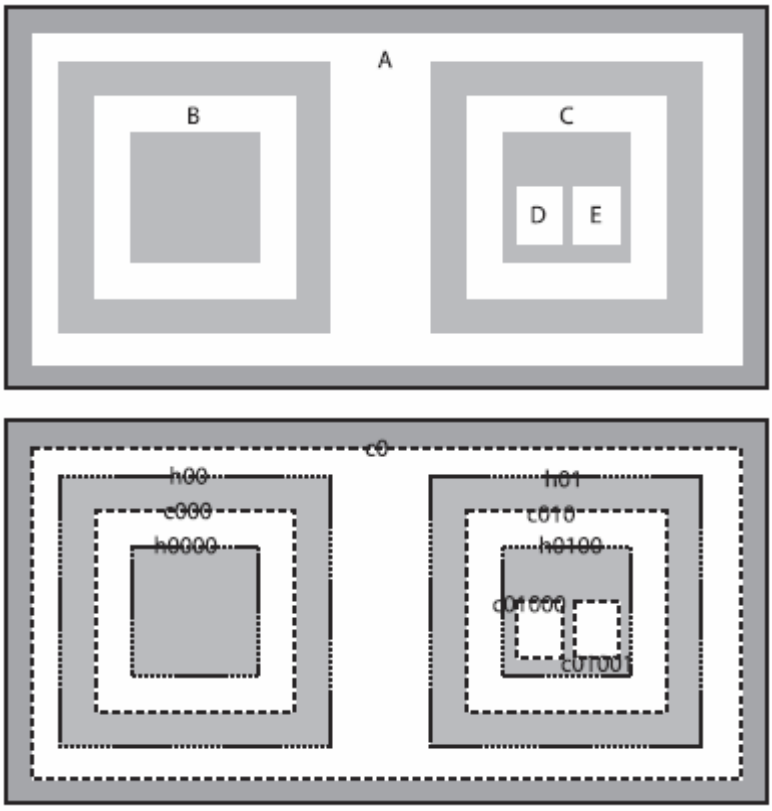
Voimme kuvasta 5 ja 6 nähdä näiden kahden eri operaation selkeän eron.

3.4 Ääriviivat (contour)

Bradsky ja Kaehler (2008) määrittävät ääriviivat seuraavasti:

> *Ääriviivat* ovat lista pisteitä, jotka määrittävät kuvassa olevia käyriä. Tämä esitystapa voi vaihdella riippuen tilanteesta, ja on monia tapoja joilla käyrän voi esittää. Ääriviivat on esitetty OpenCV:ssä sarjoina, joissa jokainen sarjan jäsen sisältää tiedon seuraavan käyrän pisteen sijainnista. [1, s. 234.]

OpenCV sisältää suoraan funktion, jolla näitä voidaan etsiä kuvasta, *findContours.* Tämä funktio voi käyttää kuvia, jotka on luotu *canny*-, *threshold* tai *adaptiveThreshold* funktioita käyttäen. Käytännössä kuvassa tarvitsee siis olla positiivisen ja negatiivisen alueen raja implisiittinen, tai canny-funktion tapauksessa, raja-pikseleitä alueiden välillä. Seuraavassa kuvassa 7 havainnollistetaan findContoursin toimintaa. Ylempi kuva näyttää kuvan ennen findContoursia, ja alempi kuva näyttää löydetyt ääriviivat.



Kuva 7. FindContours-funktion tulos [1, 235].

Löydetyt ääriviivat voivat olla kahta eri tyyppiä: Ulkoisia ääriviivoja (katkoviivat) tai reikiä (pisteviivat). OpenCV:llä tehty C++ *findContours*-funktio on muotoa

```
void findContours(InputOutputArray image, OutputArrayOfArrays contours,
OutputArray hierarchy, int mode, int method, Point offset=Point());
```

Listaus 1. findContours-funktio.

Tässä funktiossa **image** on lähdekuva, joka on 8-bittinen yksikanavainen kuva, esimerkiksi harmaasävykuva tai binäärikuva. **Contours** taas tarkoittaa löydettyjä ääriviivoja, eli mihin muuttujaan löydetyt ääriviivat talletetaan. **Hierarchy** on valinnainen ulostulo-

vektori, joka sisältää tietoa kuvan topologiasta. **Mode** tarkoittaa, mitä ääriviivojen hakumetodia käytetään ääriviivojen kartoituksessa, jotka käydään läpi myöhemmin. **Method** on ääriviivojen arviointimetodi, joka käydään myös läpi myöhemmin. **Offset** on valinnainen arvo, jonka mukaan jokaista ääriviivan pistettä voidaan siirtää.

Modella on neljä erilaista optiota, jotka löytyvät OpenCV:n dokumentaatiosta [18]:

CV_RETR_EXTERNAL

Tämä optio palauttaa vain uloimmat ääriviivat.

CV_RETR_LIST

Tämä optio palauttaa kaikki ääriviivat luomatta niille hierarkiaa. Tämä on yksinkertaisin tapa saada kuvasta kaikki ääriviivat.

CV_RETR_CCOMP

Tämä optio palauttaa kaikki ääriviivat ja luo niille kaksitasoisen hierarkian. Korkeammalla tasolla sijaitsevat komponenttien uloimmat ääriviivat, ja alemmalla tasolla sijaitsevat reikien ääriviivat. Jos reikä yhdistyy ulompaan ääriviivaan, sijaitsee se korkeammalla tasolla.

CV_RETR_TREE

Tämä optio palauttaa kaikki ääriviivat ja rakentaa niille koko hierarkian sisäkkäisistä ääriviivoista.

Methodilla on kolme erilaista optiota, jotka myöskin löytyvät OpenCV:n dokumentaatiosta [18]:

CV_CHAIN_APPROX_NONE

Tämä optio säilöö jokaisen ääriviivapisteen. Mitkä tahansa kaksi peräkkäistä ääriviivan pistettä, esimerkiksi (x1, y1) ja (x2, y2), ovat joko horisontaalisia, vertikaalisia tai vinottaisia naapureita.

CV_CHAIN_APPROX_SIMPLE

Tiivistää horisontaaliset, vertikaaliset sekä diagonaaliset osiot, ja jättää jäljelle vain niiden loppupisteet. Esimerkiksi neliönmuotoinen ääriviiva on muodostettu vain neljästä pisteestä.

CV_CHAIN_APPROX_TC89_L1 ja **CV_CHAIN_APPROX_TC89_KCOS**

Käyttää yhtä *Teh-Chin*-ketjun arviointialgoritmia ääriviivojen arviointiin [19].

3.5 Pikseliyhteydet (Pixel Connectivity)

Pikseliyhteys tietokonenäön tapauksessa tarkoittaa, millä tavoin pikselit ovat 2d-kuvissa liittyneet naapureihinsa. Jotta kaksi pikseliä voisivat olla yhteydessä, täytyy niiden arvojen olla samat binäärikuvassa, tai erittäin lähellä samaa harmaasävykuvissa. Esimerkiksi binäärikuvassa yhteydessä olevien pikseleiden arvot tulee olla tasan sama arvo, esim. {1}. Harmaasävykuvien tapauksessa yhteydessä olevien pikseleiden arvot pitää olla samalta harmaasävyn alueelta, esim. {22, 23, 24,...,40} [20].

Taulukko 3. 4-yhteyksinen pikselitaulu

0	1	0	0
1	1	1	0
0	1	0	0
0	1	0	0

4-yhteyksiset pikselit ovat naapureita jokaiselle pikselille, joka koskee yhtä niiden reunoista. Pikselit ovat yhteydessä horisontaalisesti ja vertikaalisesti kuten taulukossa 3.

Taulukko 4. 6-yhteyksinen pikselitaulu

0	1	0	0
1	0	1	0
0	1	0	0
0	0	0	0

6-yhteyksiset pikselit ovat naapureita jokaiselle pikselille, joka koskee yhtä niiden kulmista. Pikselit ovat siis diagonaalisesti yhteydessä toisiinsa kuten taulukossa 4.

Taulukko 5. 8-yhteyksinen pikselitaulu

1	1	0	0
1	1	0	0
0	1	1	0
0	0	1	0

8-yhteyksiset pikselit ovat naapureita jokaiselle pikselille, joka koskee yhtä niiden kulmista tai reunoista. Pikselit ovat siis diagonaalisesti, horisontaalisesti tai vertikaalisesti yhteydessä toisiinsa kuten taulukossa 5.

3.6 Yhdistetty komponentti (connected component)

Yhdistetyt komponentit ovat suuria diskreettejä alueita, joiden pikselien arvot tai värit ovat erittäin lähellä toisiaan [1, s. 117]. Esimerkiksi iso alue, jossa jokainen pikseli on toisensa naapuri, ja ovat täten yhdistettyjä pikseleitä, on yhdistetty komponentti.

3.7 Blob

Blob (Binary Large Object) on kokoelma binääridataa, joka on talletettu yhteen kokonaisuuteen. Tietokonenäössä tämä tarkoittaa jotakin tiettyä kuvan osaa, joka on erotettu muusta kuvasta omaksi kokonaisuudekseen, blobiksi.

4 Objektin ja muodon tunnistus

Tässä luvussa käsittelen, miten ohjelma voidaan opettaa tunnistamaan tiettyjä objekteja kuvasta tai videolta opetuksen avulla. Käsittelen pääosin projektissa käytettyä Haar-koulutusta, muita OpenCV:n tukemia *koneoppimisen* algoritmeja käsittelemme myöhemmässä osuudessa.

4.1 Haar-koulutus, mitä se on ja miten se toimii.

Haar-koulutus jota tässä työssä käytetään, käyttää hyväkseen *Haar-tyyppisiä piirteitä*, joita käytetään objektin tunnistuksessa. Näistä Haar-tyyppisistä piirteistä voimme rakentaa OpenCV:n avulla *kaskaadiluokittelijan* (cascade classifier), johon on opetettu jokin tietty objekti käyttäen monia positiivisia ja negatiivisia kuvia. Tätä tekniikkaa käytetään usein kasvojen tunnistukseen, mutta sitä voidaan myös käyttää minkä tahansa muun opetetun objektin tunnistukseen.

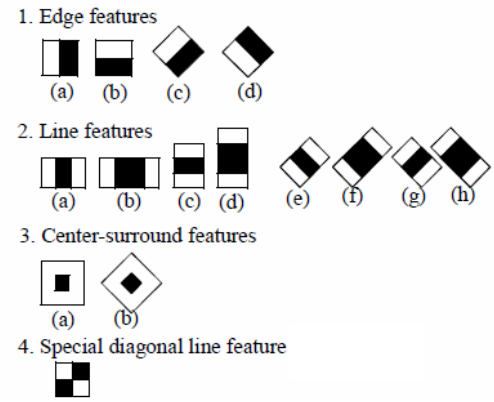
Haar-tyyppiset piirteet

Piirteiden laskenta kuvista käyttäen kuvan pikseleiden intensiteettiä (RGB) on erittäin vaativaa toimintaa tietokoneelta, joten sitä parempi keino piti etsiä käyttäen jotain muuta tekniikka. Papageorgioun, Orenin ja Poggion julkaisussa [21] ehdotettiin käytettä-

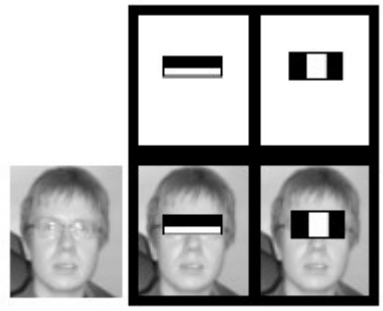
väksi Haar-aallokemuunnosta pikseleiden intensiteetin sijasta, ja Paul Viola ja Michael Jones käyttivätkin tätä tekniikkaa, ja kehittivät sitä käyttäen niinkutsutut Haar-tyyppiset piirteet. Viola-Jones kertovat näiden piirteiden käytöstä seuraavasti:

> Käyttämämme objektin havaitsemisprosessi luokittaa kuvat niiden sisältämien yksinkertaisten piirteiden arvojen mukaan. On monta syytä käyttää piirteitä eikä suoraan pikseleitä. Yleisin syy on, että piirteet voivat koodata improvisoitua tietotaitoa joka on vaikeaa oppia rajatulla määrällä koulutusdataa. On myös toinen kriittinen motivaatio piirteiden käyttämiseen: piirteitä käyttävä järjestelmä toimii paljon nopeammin kuin pikseleitä käyttävä järjestelmä. Yksinkertaiset piirteet ovat samankaltaisia kuin Haarin perusfunktiot, joita Papageorgiou ym. käyttivät [21].
>
> Käytämme kolmenlaisia piirteitä. *Kaksoisneliöpiirteen* (two-rectangle feature) arvo on näiden kahden neliömäisen alueen väliin jääneiden pikseleiden summa. Alueilla on sama koko ja muoto, ja ovat vaaka- tai pystysuunnassa vierekkäiset. *Kolmoisneliöpiirre* (three-rectangle feature) summaa kahden uloimman neliön alueet yhteen ja tämä summa vähennetään keskimmäisen neliön summasta. *Neljäneliöpiirre* (four-rectangle feature) laskee erotuksen vinottain vastakkaisten neliöiden välillä. [14, s 2.]

Haar-tyyppiset piirteet käsittelevät vierekkäisiä nelikulmaisia alueita havaintoikkunassa, summaavat niiden alueiden pikselien intensiteetin ja laskevat näiden alueiden summien erotuksen. Tätä erotustä käyttäen kuvan alaosiot voidaan kategorisoida. Kasvojen tunnistuksen tapauksessa voimme siis kategorisoida silmät Haar-piirteeksi, koska voimme jo katsomalla päätellä että silmät ovat useimmissa tapauksissa tummempia kuin posket. Tämä yleinen Haar-piirre on kasvojentunnistuksessa siis kaksi vierekkäistä nelikulmiota silmien ja poskien yläpuolella. Näiden nelikulmioiden sijainti määräytyy havaintoikkunan, kasvojen tässä tapauksessa, suhteen kuten kuvassa 9.



Kuva 8. Esimerkkejä Haar-koulutuksen käytössä olevista piirteistä [13, s. 2]



Kuva 9. Kasvojen tunnistuksessa käytetyt nelikulmaiset piirteet ja niiden sijainnit

Objektin havaintovaiheessa kohteen, eli havaittavan objektin, kokoinen ikkuna siirretään annettuun kuvaan, ja kunkin kuvan aliosion Haar-tyyppinen piirre lasketaan. Tätä erotusta verrataan sitten opittuun kynnysarvoon, joka erottaa objektit epäobjekteista. Tämä ei kuitenkaan toimi pelkästään yhdellä Haar-tyyppisellä piirteellä, koska ne ovat heikkoja luokittajia, vaan siihen tarvitaan monia Haar-tyyppisiä piirteitä. Näistä heikoista piirteistä luodaan luokittajakaskaadi (*classifier cascade*) muodostamaan vahva luokittaja tai oppija. OpenCV:ssä on valmiina funktiot tällaisen kaskaadin luontiin ja käyttöön XML-muodossa.

Luokittajakaskaadi (Classifier Cascade)

Luokittajakaskaadi (*cascade of boosted classifiers working with haar-like features*) koulutetaan käyttäen joitakin satoja samankokoisia kuvia jostakin tietystä objektista erilaisissa taustoissa, joita kutsutaan positiivisiksi kuviksi, ja samankokoisista kuvista jotka kuvaavat joitakin ihan muita asioita, joita kutsutaan negatiivisiksi kuviksi. Eli jos haluamme havaita banaania, käytämme noin sataa kuvaa banaanista eri taustoilla ja noin sataa kuvaa jostakin aivan muusta kuin banaanista, vaikkapa pelkistä taustoista ilman banaania.

Tämä koulutetaan eri tasoissa aloittaen ensimmäiseltä tasolta ja jatkaen koulutusta käyttäjän määräämään tasoon asti. Koska jokainen seuraava taso on aiempaa tasoa vaikeampi havaita, ja jokainen lisätty taso vähentää havaintojen määrää, on löydettävä tasapaino havaintojen ja väärien havaintojen määrälle. Jos tasoja on erittäin paljon, lisää tämä koulutuksen prosessointiaikaa, ja tekee havainnoinnista erittäin tiukan, eli pienetkin erot positiivisesta kuvasta aiheuttavat havainnoinnin epäonnistumisen.

Viola ja Jones kertovat ensimmäisen tason kaskaadin rakenteesta seuraavasti:

> Kaskaadin rakenne peilaa sitä faktaa, että suurin osa yhden kuvan havaintoikkunoista ovat negatiivisia. Niinpä kaskaadi pyrkii hylkäämään mahdollisimman monta negatiivista niin aikaisella tasolla kuin mahdollista. Vaikka positiivinen tapahtuma aktivoi jokaisen kaskaadin luokittajan arvioinnin on tämä erittäin harvinainen tapahtuma. [14, s. 5.]

Suurin osa annetusta kuvasta saaduista kohdeikkunoista on negatiivisia, koska ne eivät sisällä haluttuja Haar-piirteitä, ja kaskaadin koulutuksessa nämä negatiiviset pyritään poistamaan mahdollisimman aikaisin, tässä esimerkissä ensimmäisellä tasolla.

Jokaisella kaskaadin tasolla käytetyt luokittajat ovat jo itsessään monimutkaisia, ja ne on rakennettu perusluokittajista käyttäen yhtä neljästä "boostaus" (boosting) -tekniikoista, eli painotettua äänestystä. Boostaus on voimakas opetuskonsepti, joka tarjoaa ratkaisun valvottuun luokituksen oppimistehtävään. Se yhdistää monta heikkoa luokittajaa, joiden tarkkuus tarvitsee olla vain parempi kuin satunnainen arvailu, ja tekee niistä yhden kokonaisen voimakkaan luokittajan. OpenCV tukee näistä neljää: Discrete Adaboostia (Adaptive Boosting), Real Adaboostia, Gentle Adaboostia ja Logitboostia [22]. Näiden algoritmien rakenne on erittäin lähellä toisiaan. Seuraavassa Diskreetin AdaBoostin algoritmi:

Kaksitasoinen Discrete Adaboost algoritmi

1. Aseta N esimerkit $(x_i, y_i)1N$ ja $x_i \in R^K, y_i \in -1, +1$.
2. Aseta painotukset $w_i = 1/N, i = 1, ..., N$.
3. Toista $m = 1, 2, ..., M$:

   3.1. Sovita luokittaja $f_m(x) \in -1, 1$ käyttäen painoja w_i testidataan.

   3.2. Laske $err_m = E_w[1_{(y \neq f_m(x))}], c_m = \log((1 - err_m)/err_m)$.

   3.3. Aseta $w_i \Leftarrow w_i \exp[c_m 1_{(y_i \neq f_m(x_i))}], i = 1, 2, ..., N,$ ja uudelleennormalisoi jotta $\Sigma i w_i = 1$.
4. Luokita uudet otokset *x* käyttäen kaavaa: $\mathrm{sign}(\Sigma m = 1 M c_m f_m(x))$. [23]

Discrete Adaboost - CvBoost::DISCRETE

Diskreetti AdaBoost.

Real AdaBoost - CvBoost::REAL

Käyttää luottamusmitoitettuja ennustuksia ja toimii hyvin kategorisen datan kanssa.

Gentle AdaBoost CvBoost::GENTLE

Asettaa vähemmän painoa poikkeaville havainnoille, ja on usein hyvä regressiodatan käsittelyyn.

LogitBoost - CvBoost::LOGIT.

Tämä algoritmi on myös hyvä regressiodatan käsittelyssä.

Näistä boostaus-algoritmeista useiten käytettävät ja parhaat ovat Gentle AdaBoost ja Real AdaBoost [23].

Kun kaskaadiluokittelija on koulutettu, sitä voidaan käyttää suoraan OpenCV:n avulla kuvissa oleviin alueisiin, jotka ovat samankokoisia kuin koulutuksessa käytetty havaintoikkuna. Jos alue todennäköisesti sisältää halutun objektin, luokittelijan ulostuloksi tulee "1". Jos se ei sisällä objektia, antaa se ulostuloksi "0". Koska luokittelija havaitsee vain tietyn kokoisen alueen kuvasta, ja havaintoikkuna on usein pienempi kuin koko annettu kuva, voidaan kuva "skannata" siirtämällä havaintoikkunaa kuvan sisällä ja tarkistaa täten koko kuvan alue. Luokitteljan kokoa voidaan myös muuttaa, jotta voimme löytää haluttuja objekteja riippumatta niiden koosta kuvassa. [22.]

Esimerkkejä

Näytän tässä joitakin yksinkertaisia esimerkkejä positiivisista ja negatiivisista kuvista, joilla voimme kouluttaa luokittajakaskaadin havaitsemaan jonkin tietyn objektin kuvasta. Näissä esimerkeissä pyrimme havaitsemaan avointa kämmentä.

Kuva 10. Positiivisien kuvien esimerkkejä

Kuva 11. Negatiivisten kuvien esimerkkejä

Kuvien 10 ja 11 kaltaisia kuvia käyttäen voimme siis kouluttaa kaskaadin havaitsemaan kämmeniä, jotka ovat eri kulmissa, taustoissa ja valoisuuksissa. Havainnointia tästä pystyisi vielä parantamaan käyttämällä tarkemmin rajattuja kämmeniä. Tulemme käymään tätä osiota vielä tarkemmin esimerkkiohjelmissa.

Optimaalisen havainnoinnin saamiseksi tulee positiiviset ja negatiiviset kuvat ottaa samalla kameralla kuin niitä yritetään havaita, koska kynnysarvot ja haluttu tarkkuus muuttuvat paljon erilaatuisilla kameroilla. Jos luokittelija on koulutettu käyttäen 240p:n videokameraa, ja arvot ja tarkkuus säädetty tätä kameraa käyttäen, on oikean havainnoinnin saaminen 720p:n kameralla vaikeampaa. Sama pätee siirryttäessä 720p:n kamerasta 240p:n kameraan. Positiivisten ja negatiivisten kuvien määrä vaikuttaa merkittävästi havainnointiin, ja riippuen halutusta tarkkuudesta ja objektin monimutkaisuudesta tämä määrä voi kasvaa tai laskea. Parhaimman tarkkuuden saa käyttämällä suurta määrää esimerkkikuvia, 1000 ja enemmän. Suuren kuvamäärän käyttäminen kuitenkin lisää koulutukseen kuluvaa aikaa.

4.2 OpenCV:llä tapahtuva kuvien kouluttaminen

OpenCV tarjoaa kaksi sovellusta kaskaadiluokittelijan koulutukseen, joita ovat *opencv_haartraining* ja *opencv_traincascade*. Näistä kahdesta sovelluksesta *opencv_traincascade* on uudempi ja on kirjoitettu C++:llä, mutta suurin ero näiden kahden kesken on se, että *opencv_traincascade* tukee Haar- sekä LBP (Local Binary Patterns)-piirteitä. [24.] Näiden kahden sovelluksen lisäksi OpenCV tarjoaa kaksi apuohjelmaa koulutuksen helpottamiseksi. Ne ovat *opencv_createsamples* sekä *opencv_performance.* Näistä kahdesta käytämme tässä työssä *opencv_createsamples*-apuohjelmaa, ja saamme sitä käyttäen vektoritiedoston joka sisältää positiivisten kuvien datan, jota voi suoraan käyttää *opencv_haartraining*- sekä *opencv_traincascade* -sovelluksissa. *Opencv_performancella* voimme arvioida luokittajien laatua, mutta sitä voidaan käyttää vain *opencv_haartraining*-sovelluksen kanssa. Koska jotkin hyödylliset kirjastot ja työkalut, kuten edellämainittu *opencv_performance*, eivät toimi uudemman *opencv_traincascaden* kanssa, niin käytämme tässä työssä myös vanhempaa *opencv_haartrainingia*. Pääosin käytämme työssä kuitenkin *opencv_traincascadea*, sillä se tukee monisäikeisyyttä.

Negatiivisten esimerkkien koulutus

Negatiiviset esimerkit, eli kuvat, joissa ei ole haluttua objektia, pitää hankkia ja järjestää manuaalisesti. Niiden luontiin ei ole olemassa apuohjelmaa, mutta koska niiden ainoa kriteeri on, että ne ovat suuremman kokoisia kuin käytettävä havaintoikkuna, on niiden hankinta suhteellisen helppoa. Tässä työssä käytetään negatiivisina esimerkkeinä suoraan videokamerasta kaapattua kuvaa, jossa jokaisesta kameran framesta on luotu oma erillinen kuva. Tällä metodilla saamme helposti monia kuvia. Manuaalista siivoamista tarvitaan usein silti, koska nopeat liikkeet aiheuttavat kuvaan monia häiriöitä, ja kuvat, joissa näitä häiriöitä on, kannattaakin poistaa.

Positiivisten esimerkkien koulutus

Positiiviset esimerkit luodaan *opencv_createsamples*-apuohjelmalla, ja ne voidaan luoda käyttäen pelkästään yhtä kuvaa tai kokoelmasta aiemmin merkattuja kuvia. Tämä apuohjelma vain muokkaa objektin perspektiiviä, joten positiivisia kuvia tulee olla paljon ennen niiden antamista tälle apuohjelmalle, varsinkin monimutkaisempien objektien tapauksessa. Helpon kaksiulotteisen kuvan tapauksessa vain yksi positiivinen esimerkki voi olla tarpeeksi, koska niissä perspektiivin muutos on aivan tarpeeksi. Kolmiulotteis-

ten objektien ja muotojen tapauksessa pelkkä perspektiivin muutos ei riitä tunnistuksessa. Kasvojentunnistuksen tapauksessa suositeltu määrä ennen niiden ajamista *opencv_createsamplesin* läpi onkin tuhansissa, koska tunnistuksessa pitää ottaa huomioon ikäerot, kasvojen värierot, parrat sekä kasvoilla näkyvät tunteet.

Apuohjelma muokkaa sille annettuja positiivisia esimerkkikuvia kääntämällä niitä eri suuntiin, vaihtamalla kuvan intensiivisyyttä sekä asettamalla kuvia eri taustoihin. Voimme kontrolloida muutosten määrää komentoriviargumentein [24]. Näillä komentoriviargumenteilla voidaan nimetä tuloksena syntyvä vektoritiedosto, määrittää haluttu lähdekuva, lista taustakuvista, asettaa haluttu esimerkki-kuvien määrä, vaihtaa taustan väri (normaalisti harmaasävy), määrätä kynnysarvot, asettaa käänteiset värit ym. määritteitä, jotka näemme listauksessa 2.

```
-vec <vec_file_name>                          – Nimeää tuloksena syntyvän vektoritiedoston
-img <image_file_name>                        – Lähdekuvan asetus
-bg <background_file_name>                    – Lista, joka sisältää taustakuvat
-num <number_of_samples>                      – Haluttu positiivisten esimerkkien määrä
-bgcolor <background_color>                   – Taustan väri
-bgthresh <background_color_threshold>        – Taustan kynnysarvon asetus
-inv                                          – Värien kääntäminen
-randinv                                      – Värien satunnainen kääntäminen
-maxidev <max_intensity_deviation>            – Kuvan etuosan pikselien maks. voimakkuusero
-maxxangle <max_x_rotation_angle>             – X-akselin maks. kierto radiaaneissa
-maxyangle <max_y_rotation_angle>             – Y-akselin maks. kierto radiaaneissa
-maxzangle <max_z_rotation_angle>             – Z-akselin maks. kierto radiaaneissa
-show                                         – Debug-optio. Näyttää jokaisen esimerkin jos
                                              asetettu
-w <sample_width>                             – Tuloskuvien leveys
-h <sample_height>                            – Tuloskuvien korkeus
```

Listaus 2. Createsamples-apuohjelman komentoriviparametrit [24]

Käytössämme on siis varsin paljon erilaisia asetuksia, joita voimme säätää luodessamme positiiviset esimerkkimme. Itse kaskaadin koulutuksessa on myös käytössä lista omia komentoriviargumentteja.

Kaskaadin koulutus

Esimerkkikuvien luonnin jäkeen voimme aloittaa itse kaskaadin koulutuksen, jossa käytämme *opencv_haartraining*-sovellusta. Kuten aiemmin on mainittu, tämänkin sovelluksen käytössä on komentoriviargumentteja listauksen 3 mukaan.

```
-data <cascade_dir_name>                          – Koulutetun kaskaadin sijainti
-vec <vec_file_name>                              – Positiiviset kuvat sisältävä vektoritiedosto
-bg <background_file_name>                        – Taustan sisältävä tiedosto
-npos <number_of_positive_samples>                – Positiivisten kuvien määrä
-nneg <number_of_negative_samples>                – Negatiivisten kuvien määrä
-nstages <number_of_stages>                       – Kuinka monta kaskaaditasoa käytetään
-nsplits <number_of_splits>                       – Jakautumisten määrä
-mem <memory_in_MB>                               – Koulutukseen käytettävän RAM-muistin määrä
-sym (default)                                    – Käytetään jos haluttu objekti on symmetrinen,
                                                    muuten nonsym
-minhitrate <min_hit_rate>                        – Luokittajan pienin toivottu osumatarkkuus
-maxfalsealarm <max_false_alarm_rate>             – Luokittajan suurin toivottu väärien hälytysten
                                                    määrä
-weighttrimming <weight_trim_rate>                – Käytetäänkö kuvien trimmausta, ja kuinka
                                                    vahvasti
-eqw
-mode <BASIC (default) | CORE | ALL>              – Minkätyyppisiä Haar -piirteitä käytetään
                                                    koulutuksessa
-w <sampleWidth>                                  – Esimerkkikuvien leveys
-h <sampleHeight>                                 – Esimerkkikuvien korkeus
-bt <{DAB, RAB, LB, GAB(default)}>                – Boostatun luokittajan tyyppi
-err <misclass (default)|gini|entropy>
-maxtreesplits <max_number_of_splits_in_tree_cascade>   – Puurakenteen
                                                    jakautumisten maks. määrä
-minpos <min_number_of_positive_samples_per_cluster>    – Positiivisten kuvien min.
                                                    määrä per klusteri
```

Listaus 3. Haartrainingin komentoriviargumentit [15]

Koska käytämme tässä työssä kumpaakin *opencv_haartraining*- sekä *opencv_traincascade* -ohjelmaa, voimme listauksesta 4 nähdä jälkimmäisen komentoriviargumentit. Argumentit ovat varsin samannäköisiä *opencv_haartraining*-ohjelman kanssa.

```
-data <cascade_dir_name>                          – Koulutetun kaskaadin sijainti
-vec <vec_file_name>                              – Positiiviset kuvat sisältävä vektoritiedosto
-bg <background_file_name>                        – Taustan sisältävä tiedosto
-numPos <number_of_positive_samples>              – Positiivisten kuvien määrä
-numNeg <number_of_negative_samples>              – Negatiivisten kuvien määrä
-numStages <number_of_stages>                     – Kuinka monta kaskaaditasoa käytetään
-precalcValBufSize <precalculated_vals_buffer_size_in_Mb>   – Esilaskettujen piirteiden
                                                    arvojen puskuri
-precalcIdxBufSize <precalculated_idxs_buffer_size_in_Mb>   – Esilaskettujen piirteiden
                                                    indeksien puskuri
```

-baseFormatSave	– Kaskaadi talletetaan vanhassa formaatissa jos asetettu
-stageType <BOOST(default)>	– Kaskaaditasojen tyypit
-featureType<{HAAR(default), LBP}>	– Mitä piirretyyppiä käytetään
-w <sampleWidth>	– Esimerkkikuvien leveys
-h <sampleHeight>	– Esimerkkikuvien korkeus
-bt <{DAB, RAB, LB, GAB(default)}>	– Boostatun luokittajan tyyppi
-minHitRate <min_hit_rate>	– Luokittajan pienin toivottu osumatarkkuus
-maxFalseAlarmRate <max_false_alarm_rate>	– Luokittajan suurin toivottu väärien hälytysten määrä
-weightTrimRate <weight_trim_rate>	– Käytetäänkö kuvien trimmausta, ja kuinka vahvasti
-maxDepth <max_depth_of_weak_tree>	– Heikon puun maksimisyvyys
-maxWeakCount <max_weak_tree_count>	– Heikkojen puiden maksimimäärä per taso
-mode <BASIC (default) | CORE | ALL>	– Mitä Haar -tyyppisiä piirteitä koulutuksessa käsitellään. BASIC käyttää vain pystyssä olevia piirteitä, kun taas ALL käyttää myös 45-asteen kulmassa olevia käännettyjä piirteitä

Listaus 4. Traincascaden komentoriviargumentit [24]

Ohjelman suorituksen jälkeen koulutettu kaskaadi talletetaan .xml-tiedostona kansioon, joka määrättiin komentoriviargumentilla suoritusta ennen. Ennen koulutusta tulee kuitenkin ottaa huomioon siihen kuluva aika: Suurilla määrillä positiivisia ja negatiivisia esimerkkikuvia koulutus voi helposti kestää päivän tai kauemminkin. *Opencv_haartraining*-ohjelman *-mem*-option muokkaus ei nopeuta koulutusta merkittävästi, koska se vaikuttaa vain ohjelman alkulaskentaan. *Opencv_traincascaden* puskuriarvojen asetus voi nopeuttaa koulutusta jonkin verran, mutta se kestää silti paljon aikaa. Koulutus kannattaakin usein tehdä varalla olevalla koneella, kuten itse aion tehdä, tai jollakin virtuaalikoneella.

Kaskaadin koulutusvaiheessa hyvä tasojen määrä on noin 20, sillä hirvittävän suuren tasomäärän asettaminen ei paranna havainnoinnin tarkkuutta: Jos koulutusohjelma pääsee haluttuun *-minhitrate*-option määräämään tarkkuuteen ennen kaikkien tasojen käymistä läpi, tai kaikki esimerkkikuvat hylättiin, voi ohjelma päästä loppuun käymättä kaikkia tasoja läpi. Jälkimmäisessä tapauksessa esimerkkikuvien määrää tulisi kasvattaa.

```
opencv_traincascade -data luokittaja -vec esimerkit.vec -bg
negatiiviset.txt -numStages 20 -minHitRate 0.998 -maxFalseAlarmRate 0.5
-numPos 500 -numNeg 400 -w 20 -h 20 -mode ALL -precalcValBufSize 1024\
-precalcIdxBufSize 1024
```

Listaus 5. Esimerkki traincascaden käytöstä 500 positiivisella ja 400 negatiivisella esimerkkikuvalla

Käytämme tässä työssä edelläolevien apuohjelmien lisäksi Naotoshi Seon [15] luomia apuohjelmia *mergevec.cpp* sekä *createtrainsamples.pl,* joiden avulla voimme yhdistää luotuja vektoritiedostoja sekä toistaa tämän proseduurin monta kertaa. Negatiivisia ja positiivisia esimerkkikuvia voi helposti tulla tuhansia, joten tämän proseduurin tekeminen manuaalisesti veisi aivan liikaa aikaa.

Jos käytämme vanhempaa *opencv_haartraining*-ohjelmaa, voimme käyttää apuohjelmaa *opencv_performance* jonka toimii seuraavasti:

> Havainnoinin aikana havaintoikkunaa siirrettiin pikseli pikseliltä kuvan yli eri kokoisena. Ikkunan koon muutos aloitettiin alkuperäisestä koosta ja piirteitä kasvatettiin 10% ja 20% (tarkoittamaan koon kertoimia 1.1 ja 1.2) kunnes oli jossakin suunnassa suurempi kuin koko kuva. Usein useita naamoja havaittiin lähellä oikean naaman sijaintia ja kokoa. Tästä johtuen useat havaintotulokset yhdistettiin yhdeksi havainnoksi. Vastaanottavat Operatiiviset Käyrät (Receiver Operating Curves, ROC) rakennettiin vaihtelemalla vaadittua kasvojen havaintojen määrää per oikea kasvo ennen yhdeksi havainnoksi yhdistämistä. Testien aikana vain yhtä parametria muutettiin kerrallaan. Parhaat parametrit löydettyä käytettiin tätä tulosta seuraavissa testeissä. [25.]

Voimme tätä apuohjelmaa käyttäen siis arvioida luomamme luokittajakaskaadin tarkkuutta testaamatta itse kameran kautta. Tällä apuohjelmalla saamme siis tarkan tiedon, kuinka hyvin kouluttamamme kaskaadi toimii, mutta todellinen tarkkuus selviää vasta kameran kautta. Jos saatujen osumien määrä on erittäin alhainen, kameralla ei kuitenkaan kannata edes koittaa. Koska kaskaadin kouluttaminen vie paljon aikaa, koulutuksen tulokset ja testaus on tarkemmin testiohjelmissa.

Facedetect

OpenCV:n mukana tulee ohjelma *facedetect,* joka sisältää kaikki ominaisuudet kaskaadikoulutuksesta saadun .xml-tiedoston käyttöönottoon. Ohjelmaa käyttäen voimme suoraan käyttää kouluttamaamme kaskaadia havaitsemaan suorasta videokuvasta

kouluttamiamme objekteja. Koska tämän työn tarkoitus ei ole itse tämän ohjelman rakentaminen, vaan pääpaino on objektin kouluttaminen Haar-koulutuksella, käytämme *facedetect*-ohjelmaa kaikissa seuraavan kappaleen testeissä.

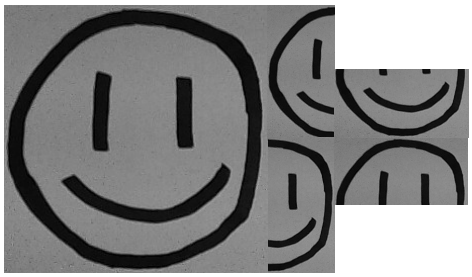
5 Testikoulutuksia ja niiden tekemiseen ja toimintaan liittyviä asioita

Tässä kappaleessa tulemme käymään Haar-koulutusta läpi tarkemmin mm. testiohjelmilla, joilla voimme arvioida kuinka hyvin objektin tunnistus toimii tällä metodilla. Tulemme myös käymään läpi tarkemmin, miten erilaiset komentorivioptiot vaikuttavat koulutukseemme.

5.1 Koulutukseen liittyviä ongelmia

Koska koulutus tapahtuu vain käyttämiämme alkuperäiskuvia ja niistä muodostettuja rajoitettuja esimerkkikuvia käyttäen, voimme saada eteemme monenlaisia ongelmia: Jos videokuvassa oleva objekti, jota koitamme havaita, on puoliksi varjossa tai muuten peitossa, mutta mikään alkuperäis- tai esimerkkikuva ei ole samantyyppisesti varjossa tai peitossa, ei objektia usein havaita. Jos objekti on vinossa, eikä se ole missään kuvissamme samantyylisesti vinossa, ei objektia havaita. Jos objekti on aivan eri valossa, esimerkiksi huone on pimeämpi, ei objektia havaita. On siis olemassa monenlaisia tilanteita, joissa objektia ei havaita, jos käytännön tilanne on erilainen testikuviemme tilanteista. *Createsamples*-apuohjelma auttaa paljon näihin tilanteisiin, sillä voimme sitä käyttäen määrittää sillä luotujen esimerkkikuvien kiertokulmat ja valaistuksen intensiivisyyden. Tämä ei kuitenkaan auta kaikissa tilanteissa, ja erittäin monien variaatioiden teko tällä ohjelmalla pidentää merkittävästi koulutukseen kuluvaa aikaa.

Usein onkin parasta käyttää monia alkuperäiskuvia, vaikka objekti olisikin erittäin yksinkertainen, kuvan 12 mukaisesti.



Kuva 12. Esimerkki kuinka yksinkertainen kuva jaetaan paloihin

Jakamalla kuva osiin voimme ehkäistä tilanteita, joissa objekti olisi puoliksi peitossa, ja tämän takia mahdoton havaita. Kuvan 12 mukainen yksinkertainen jako kuitenkin jo nelinkertaistaa esimerkkikuviemme määrän, ja täten lisää koulutusaikaa runsaasti. Tämäntyyliset yksinkertaiset kuvat on mahdollista havaita optimaalitilanteissa pelkästään jo yhden alkuperäiskuvan avulla, joten monien alkuperäiskuvien käyttämisen hyödyllisyys riippuu täysin tilanteesta, missä havainnointia tarvitaan. Kuvan osiin jakaminen aiheuttaa myös muita ongelmia, joten emme sitä tee.

Ongelmaksi voidaan myös laskea halutusta tarkkuudesta johtuva positiivisten esimerkkien määrän laskeminen: Koska esimerkkikuvat sisältävässä .vec-tiedostossa on kaikki esimerkkikuvat, emme voi suurilla minhitrate-tarkkuuksilla merkata näitä kaikkia positiivisiksi (-numpos) *traincascade*-ohjelmassa. Jos hitrate menee koulutuksen jossakin kuvassa alle asetetun minhitraten, poistetaan tämä kuva, ja ladataan .vec-tiedostosta uusi tämän tilalle. Jos kaikki .vec-tiedoston kuvat on määritetty positiivisiksi kuviksi, ei ohjelma pysty lataamaan uutta kuvaa poistetun kuvan tilalle, ja sen toiminta keskeytyy. Jos skaala on tarpeeksi pieni, kuten testikoulutuksessa 1, ei tämä välttämättä muodostu ongelmaksi.

Kuvakokoelmien määritys

Jotta pystyisimme koulutuksessa käyttämään hyväksi negatiivisia ja positiivisia kuvakokoelmia, pitää ne määrittää kokoelmatiedostoon. Tämä voi olla melkein mikä tahansa tekstitiedosto, ja tässä työssä ne on määritelty .dat-tiedostomuotoisiksi. Nämä kokoelmatiedostot sisältävät kaikki kyseisen kokoelman kuvat listauksen 6 mukaisessa muodossa.

```
[filename]
[filename]
eli esimerkiksi
img/img1.jpg
img/img2.jpg
```

Listaus 6. Kokoelmatiedoston rakenne

Kun kokoelmatiedostot on tehty, voimme niitä käyttäen käyttää kaikkia positiivisia ja negatiivisia kuvia *createsamples*-ohjelmassa, ja kaikkia negatiivisia itse *createcascade*-ohjelmassa. *Createsamples* luo meille ajamisen jälkeen kokoelmatiedoston käytössämme olevista positiivisista esimerkkikuvista .vec-muotoon, joten sitä ei meidän itse tarvitse tehdä. Jos käytössämme on vain yksi alkuperäiskuva, meidän ei tarvitse huolehtia

.vec-tiedostojen yhdistämisestä, mutta jos käytämme useampaa alkuperäiskuvaa, tulee niistä syntyvät .vec-tiedostot yhdistää esimerkiksi *mergevec*-ohjelmalla.

5.2 Kamerasta saatu kuvasta havaitseminen

Ohjelma havaitsee kamerasta määritetyn havaintoikkunan kokoisen alueen ja vertaa sitä koulutuksesta syntyneeseen .xml-kaskaaditiedostoon. Jos kamerakuvasta saadun kuvan Haar-tyyppiset piirteet vastaavat mitä havaintoikkuna sisältää kaskaaditiedostossamme, merkitsee ohjelma objektin ympyröimällä sen. Havaintoikkuna ei useinkaan ole koko kuvan kokoinen, vaan sitä siirretään läpi koko saadun kuvan, havaiten Haar-tyyppiset piirteet. Havaintoikkuna skaalautuu ohjelman suorituksen aikana itsestään, joten usein sen alkuperäinen koko ei ole hirvittävän tärkeä. Sen koon kasvattaminen voi parantaa pienten erojen havaitsemista samankaltaisista kuvista, ja käytämmekin joissakin koulutuksissa suurempaa havaintoikkunaa.

Havaitseminen aloitetaan alimmalta tasolta, joka on kaikkein sallivin, ja sen jälkeen siirrytään koko ajan vaativimpiin tasoihin. Jos jonkin tason piirteitä ei kuvassa huomata, hylätään tämä kuva kokonaan. Objekti havaitaan vain, jos siinä on koulutuksesta saadun korkeimman tason piirteet havaittuna.

5.3 Testikoulutus 1 helpolla 2d-objektilla

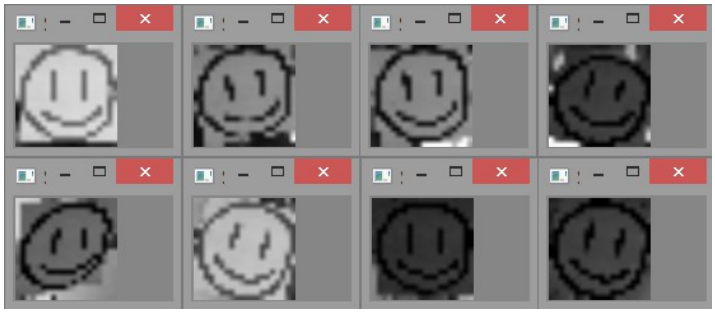
Tämä ensimmäinen kokeilu tässä työssä on tehty käyttäen kuvan 12 jakamatonta alkuperäiskuvaa, eli käytössämme oli vain yksi alkuperäiskuva koulutuksen aikana. Tästä yhdestä kuvasta teimme 1000 esimerkkikuvaa *createsamples*-ohjelman avulla, ja käytimme siinä 1000 negatiivista taustakuvaa. Käytimme seuraavaa komentoriviargumenttia luomaan alkuperäiskuvastamme haluamamme määrän esimerkkikuvia:

```
opencv_createsamples -img sample/samplecropped.bmp -num 1000 -bg
negatives2/negatives.dat -vec tester.vec -bgcolor 0 -bgthresh 0
-maxxangle  1.1 -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 20 -h
20
```

Eli loimme .vec-tiedoston tester.vec, joka sisältää 1000 alkuperäiskuvasta samplecropped.bmp tehtyä muokattua kuvaa, jotka saamme näkyviin komennolla:

```
opencv_createsamples -vec tester.vec -w 20 -h 20
```

Tämä komento näyttää jokaisen .vec-tiedostossa sijaitsevat esimerkkikuvat. Tämän testin tapauksessa se näyttäisi 1000 kuvan 13 mukaisia kuvia.



Kuva 13. Tester.vec-esimerkkikuvia

Tämä *createsamples*-ohjelman funktio aktivoituu, kun määritämme vain option -vec, emmekä lisää esimerkiksi optioita -info, -img tai -bg. [15.]

Saatuamme tämän .vec-tiedoston apuohjelmaltamme, voimme siirtyä suoraan luokittelijamme koulutukseen. Jos käyttäisimme useaa alkuperäiskuvaa, jota tulemme tekemään myöhemmissä ohjelmissa, käyttäisimme *mergevec*-apuohjelmaa yhdistääksemme jokaisesta eri alkuperäiskuvasta syntyvät .vec-tiedostot yhdeksi isoksi kokoelmatiedostoksi.

Itse kaskaadin koulutuksen teemme *opencv_createcascade*-ohjelmalla, testimme tapauksessa seuraavin optioin:

```
opencv_traincascade -data classifier -vec samples.vec -bg
negatives2/negatives.dat -numStages 20 -minHitRate 0.999
-maxFalseAlarmRate 0.5 -numPos 1000 -numNeg 1000 -w 20 -h
20 -mode ALL -precalcValBufSize 1024 -precalcIdxBufSize
1024
```

Tallennamme siis kaskaadin, ja tasoittain syntyvät .xml-tiedostot kansioon *classifier,* ja tuotamme kaskaadin käyttämällä kokoelmatiedostoja *tester.vec*, joka sisältää positiiviset esimerkkikuvamme, ja *negatives.dat*. Määritämme läpikäytävät tasot 20:een, minimiosumisen per taso 0.999 ja maksimaaliset väärät hälytykset 0.5:een, eli jos tasolla osutaan miltei 100% osumatarkkuuteen tai yli puolet havainnoista on vääriä hälytyksiä, siirrytään taso ylöspäin. Positiivisia ja negatiivisia kuvia tällä koulutuksella oli käytössä 1000 kumpaakin, ja käytössä oli kaikkien Haar-tyyppisten piirteiden tarkistukset. Kaskaadin koulutus vie paljon aikaa kuten aiemmin mainittu, varsinkin jos kaikki määritellyt

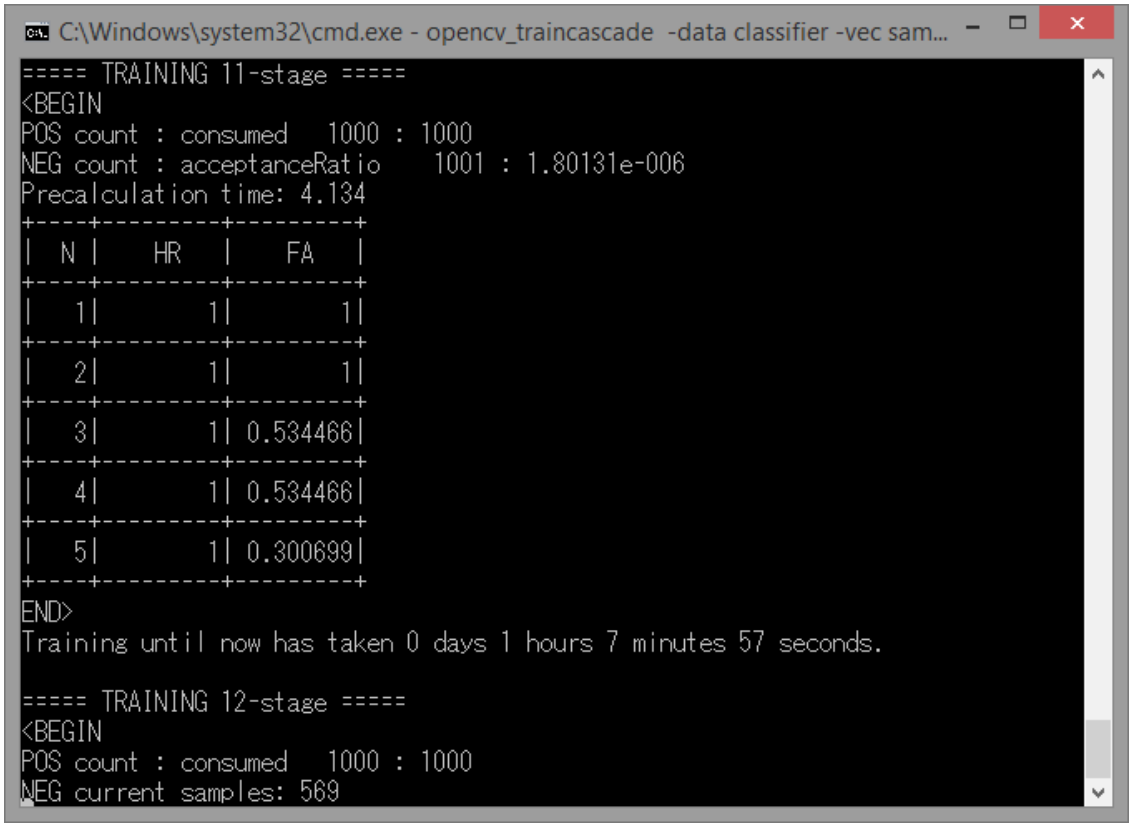
tasot käydään läpi. Kuvassa 14 näemme 11 tasoa läpikäyneen koulutuksen, joka jatkuu kolmanteentoista tasoon asti. Kolmanteentoista tasoon pääseminen vei tämän koulutuksen kanssa noin kolme tuntia, ja tämä oli käyttäen vain yhtä alkuperäiskuvaa, siitä muodostettua 1000 esimerkkikuvaa sekä 1000 negatiivista kuvaa. Tarkempi koulutus vie moninkertaisesti tämän ajan.

Kaikki testit tehtiin käyttäen *Gentle AdaBoostia*, joka on *opencv_traincascaden* oletusarvona.

Tämän ensimmäisen koulutuksen tuloksena saimme kansion classifier, joka sisältää jokaisesta tasosta luodut .xml-tiedostot, käytetyt parametrit listaava params.xml-tiedosto sekä itse cascade.xml-kaskaaditiedosto, jota käytämme objektin havaitsemiseen. Koska käytämme ohjelmaa *facedetect,* voimme aktivoida kameralla havaitsemisen komennolla

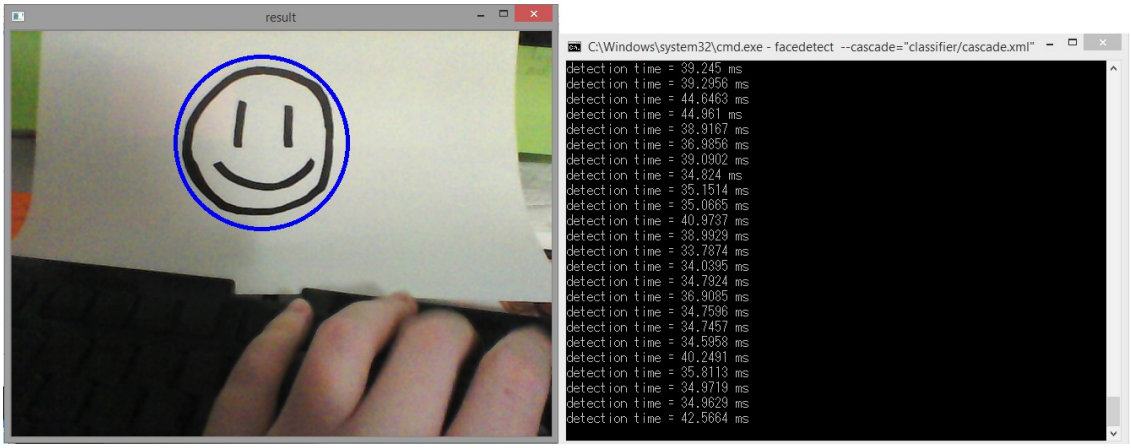
```
facedetect --cascade="classifier/cascade.xml"
```

Olettaen että koneeseen kytketty kamera voidaan havaita ohjelmalla, ja olemme oikeassa kansiossa, havaitseminen kameralla alkaa heti. Koska useimmat ohjelmat, joita käytämme sijaitsevat OpenCV:n /bin/-kansiossa, kannattaa se sijainti liittää tietokoneen PATH-muuttujaan. Jos olemme kouluttaneet kaskaadimme oikein, saamme kuvan 15 ja 16 mukaisia tuloksia kun haluttu objekti tulee kameran näkyviin. Kuviemme vasemmalla puolella näemme havaintoon kuluneen ajan, tässä tapauksessa pääsemme havaintoon noin 35 ms - 40 ms ajassa.



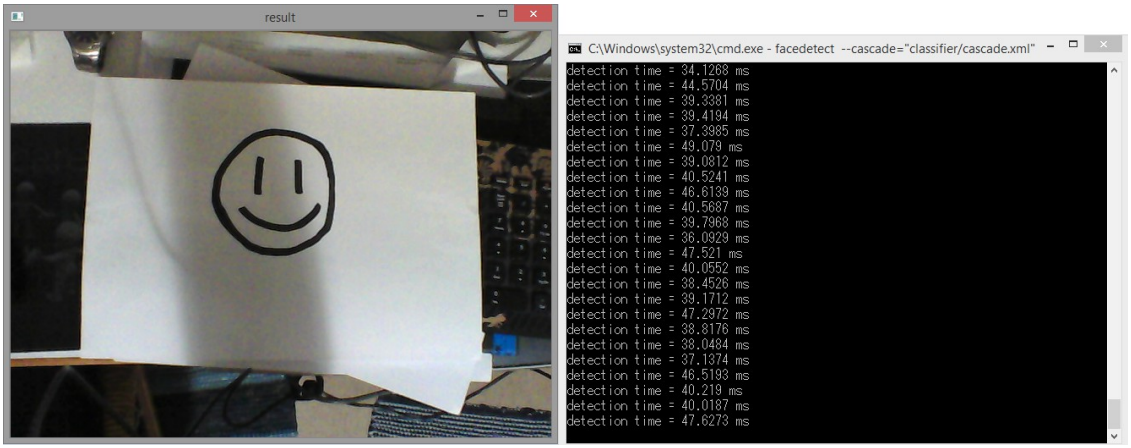
Kuva 14. 11 tasoa läpikäynyt kaskaadikoulutus

Kuva 15. Kuvan havaitseminen Haar-koulutuksen läpikäyneellä kaskaadilla

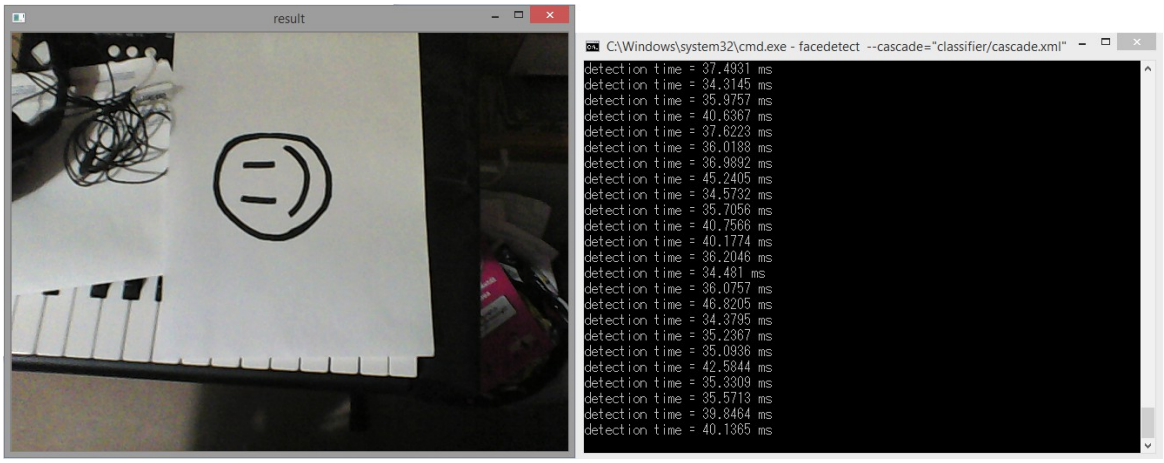


Kuva 16. Kuvan havaitseminen Haar-koulutuksen läpikäyneellä kaskaadilla

Oikein koulutetulla kaskaadilla voimme siis helposti tunnistaa hyvissä olosuhteissa olevan yksinkertaisen kuvan, mutta koulutuksessa ja esimerkkikuvien luonnissa käytetyt parametrit voivat vahvasti vaikuttaa tunnistukseen, kuten voimme huomata kuvissa 17 ja 18. Kuvassa 17 puoli kuvaa peittävän varjon aiheuttama värien voimakkuusero aiheuttaa havainnon pettämisen, eli tilanne on käytännössä sama jos kuva olisi puoliksi täysin näkymättömissä. Kuvassa 18 taas annetut kääntöparametrit eivät olleet tarpeeksi jyrkät, ja esimerkkikuvia ei tuotettu tarpeeksi, joten ohjelma ei pysty havaitsemaan kuvaa, jos se on kääntynyt jyrkemmin kuin asetetut asteet *createsamples*-ohjelmassa.



Kuva 17. Kuvan havaitsematta jääminen värien voimakkuuseron vuoksi



Kuva 18. Kuvan havaitsematta jääminen kuvan kierteisyyseron vuoksi

Kuten mainittua, käänteisyyseroista ja värien voimakkuuseroista johtuvia havainnon epäonnistumisia voimme vähentää luomalla esimerkkikuvia enemmän, ja ottamalla *createsamples*-ohjelmassa laajemman skaalan käyttöön värien voimakkuuden määräävässä optiossa sekä laajemmat kulmavalikoimat käyttöön maksimikulmat määräävässä optiossa. Kun lisäämme näitä skaaloja, joudumme heti kasvattamaan esimerkkikuvien määrää saadaksemme kaikki mahdolliset vaihtoehdot mukaan, ja kasvattamalla tästä vielä yli esimerkkien määrää, päädymme tarkempaan havainnointiin. Tämä kuitenkin kasvattaa koulutuksen aikaa merkittävästi.

Seuraavassa testiohjelmassa käytämme suurempaa skaalaa tämän saman kuvan koulutukseen, suuremmalla määrällä esimerkkikuvia sekä negatiivisia kuvia. Emme vielä tässä vaiheessa kasvata alkuperäiskuvien määrää.

5.4 Testikoulutus 2 helpolla 2d-objektilla

Tässä testissä käytimme Naotoshi Seon käyttämää negatiivisten kuvien arkistoa lisätäksemme negatiivisten kuviemme määrän 4000 kuvaan [15]. 3000 negatiivista kuvaa on yksi suositelluista määristä käytettäväksi Haar-koulutuksessa, mutta tämän määrän ylitys ei vielä haittaa. Aikavaatimusten takia käytimme tässä testissä kuitenkin vain 2000 negatiivista kuvaa näistä 4000:sta. *Createsamples*-ohjelmalla loimme 3000 esimerkkikuvaa sisältävän .vec-tiedoston seuraavin optioin:

```
opencv_createsamples -img sample/samplecropped.bmp -num 3000 -bg
test2negatives/test2negatives.dat -vec test2.vec -bgcolor 0
```

```
-bgthresh 0 -maxxangle 1.3 -maxyangle 1.3 -maxzangle 1.1
-maxidev 60 -w 20 -h 20
```

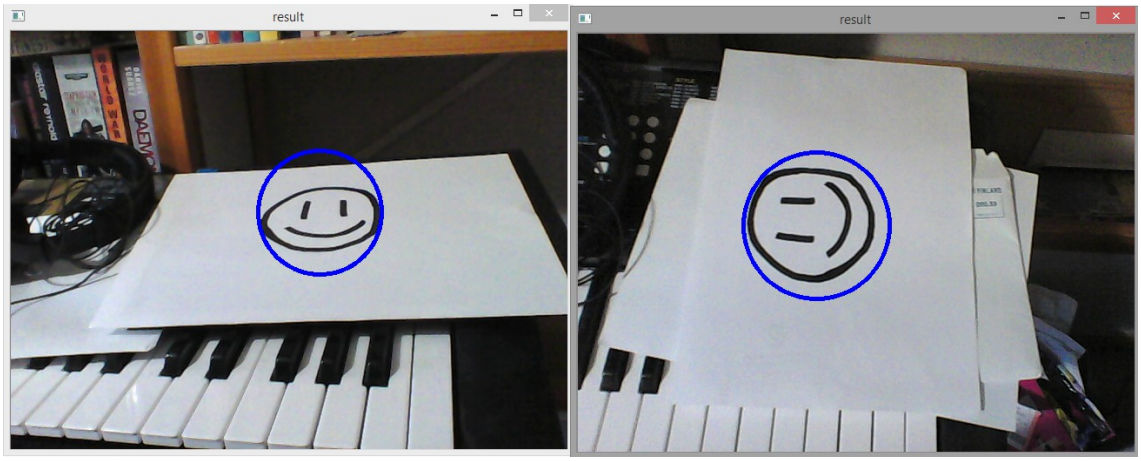
Kasvatimme tässä kiertokulmiemme määrää ja värien voimakkuuden maksimaalista erotusta. Tämän pitäisi helpottaa havaitsemista eri kulmista ja erilaisessa valaistuksessa. Esimerkkikuvien kokoelmatiedostomme sisältää nyt kuvan 19 mukaisia kuvia.

Kuva 19. Testikoulutuksen 2 esimerkkikuvia

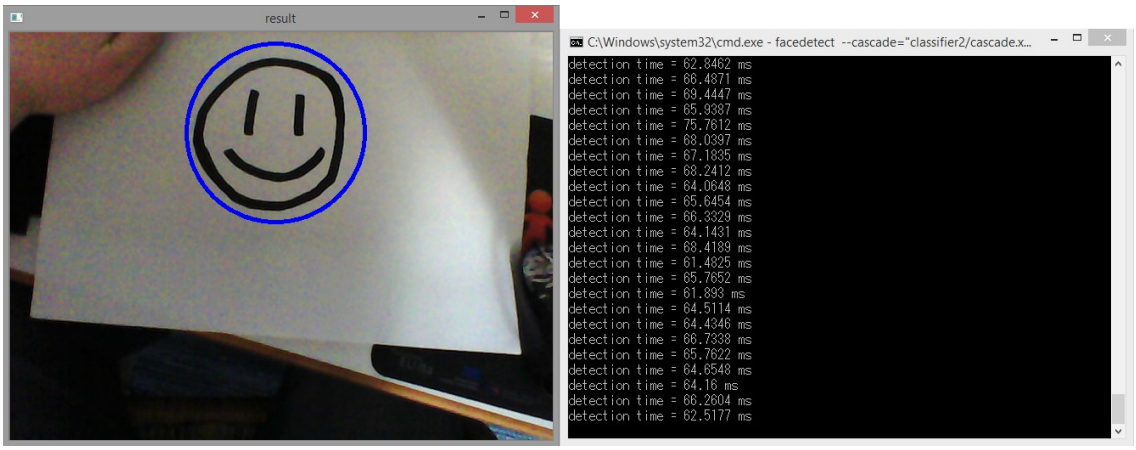
Voimme näistä esimerkkikuvista jo päätellä, että havaitseminen onnistuu suuremmilla kulmilla kuin aiemmin käyttämästämme kokoelmatiedostosta, joka oli kuvassa 13. Koulutuksen optiot ja parametrit ovat samat kuin aiemmin, mutta kuvien määrä on suurempi.

```
opencv_traincascade -data classifier -vec test2.vec -bg
test2negatives/test2negatives.dat -numStages 20 -minHitRate
0.999 -maxFalseAlarmRate 0.5 -numPos 2000 -numNeg 2000 -w 20 -h
20 -mode ALL -precalcValBufSize 1024 -precalcIdxBufSize 1024
```

Tämä koulutus kesti yhteensä noin 7 tuntia. Koska kasvatimme koulutuksen skaalaa, niin värien voimakkuuksien, kiertokulmien ja määrän mukaisesti, voimme nyt havaita objektimme paljon useammista tilanteista, kuten kuvissa 20 ja 21 näkyy. Havaintonopeus hidastuu jonkin verran edelliseen koulutukseen verrattuna, ja on nyt noin 35 ms - 60 ms.

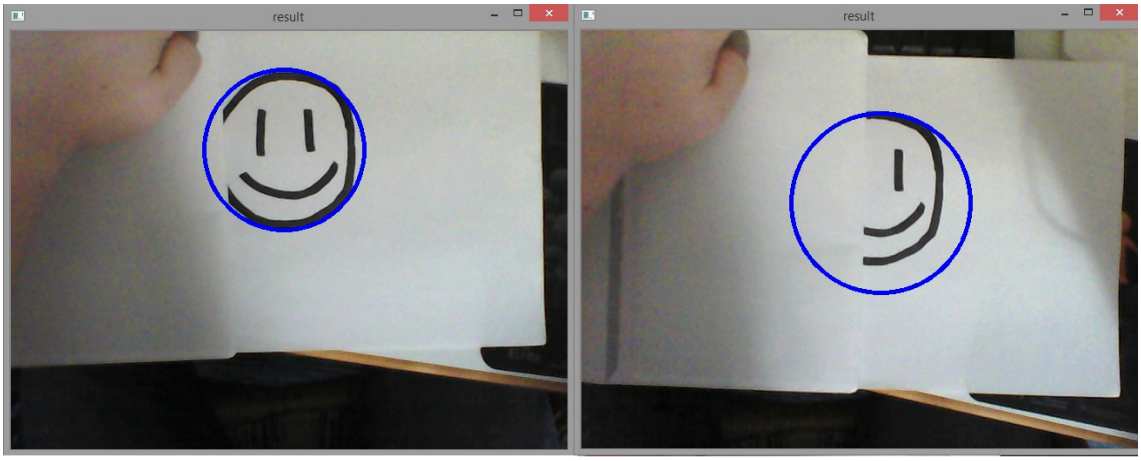


Kuva 20. Esimerkkejä suuremmasta kiertokulmasta.



Kuva 21. Esimerkki suuremmasta värien maksimivoimakkuuserosta

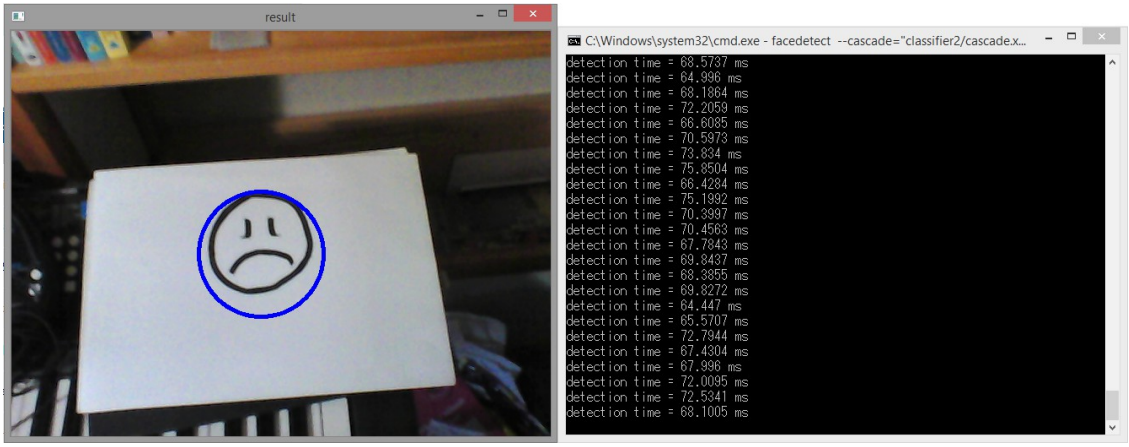
Skaalaa kasvattamalla saamme myös objektin havaittua, jos se on puoliksi peitossa, kuten kuvassa 22. Tämä johtuu lähinnä kiertokulmien kasvattamisesta ja siitä, miten Haar-tyyppiset piirteet havaitaan. Liika skaalan kasvattaminen aiheuttaa kuitenkin oman osansa ongelmia, sillä mitä suurempi skaala meillä on optioiden suhteen käytössä, sitä suurempi mahdollisuus meillä on saada vääriä havaintoja. Koulutetun objektin uniikkius ympäristönsä suhteen auttaa vähentämään näitä havaintoja.



Kuva 22. Peitetyn objektin havaitseminen. Vasemmalla testikoulutus 1, ja oikealla testikoulutus 2

5.5 Testikoulutuksen 1 ja 2 ongelmia

Suurin ongelma näissä kahdessa koulutuksessa on niitten tarkkuus: Koska käytimme vain yhtä alkuperäiskuvaa kummassakin koulutuksessa muuttaen vain koulutuksen parametreja ja skaalaa, emme voi päätyä tarkimpaan mahdolliseen tulokseen. Tällaisen yksinkertaisen hymiön havainnointi on kuitenkin todella helppoa, joten suurta määrää alkuperäiskuvia ei välttämättä tarvitakaan. Suurimmaksi ongelmaksi muodostuukin samankaltaisten objektien havainnointi, jota emme halua. Kuvasta 23 näemme, että testiohjelma havaitsee samankaltaiset, ei-halutut, objektit miltei yhtä hyvin kuin haluamamme objektit.

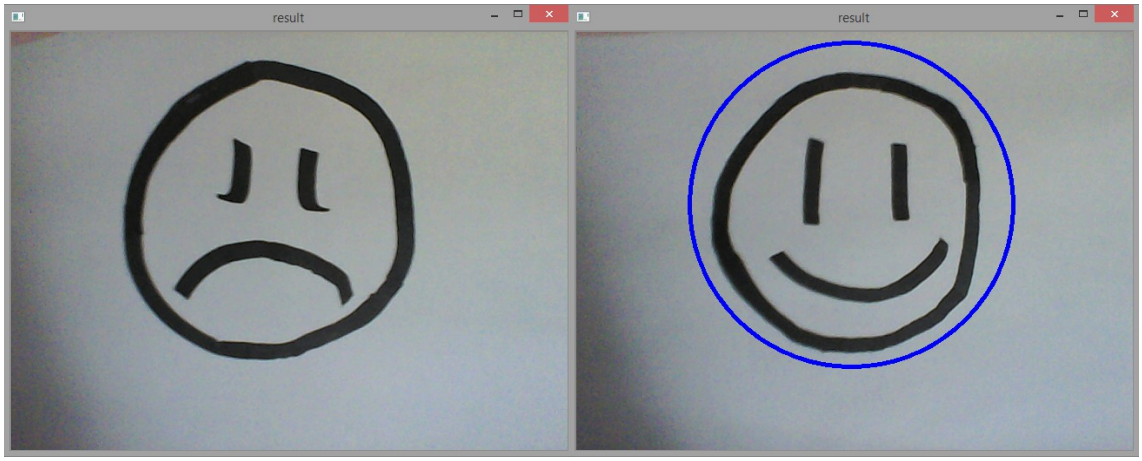


Kuva 23. Ei-haluttujen objektien havainto

Tämä tilanne toistuu kummallakin testiohjelmalla. Voimme helpottaa ongelmaa lisäämällä ei-haluttuja kuvia negatiivisten kuvien kokoelmaamme, mutta koska objekteilla on erittäin paljon samoja piirteitä, voi tämä vaikeuttaa muuta havainnointia.

5.6 Testikoulutus 3 helpolla 2d-objektilla

Tässä koulutuksessa emme muuta parametreja koulutuksesta 2, mutta lisäämme ei-haluttuja objekteja negatiivisten kuvien kokoelmaamme, ja pyrimme täten estämään niiden havaitsemisen. Lisäämme 100 kuvaa ei-halutusta objektistamme negatiivisten kuvien kokoelmaamme. Koulutuksen jälkeen saimme seuraavat tulokset ohjelmaltamme:



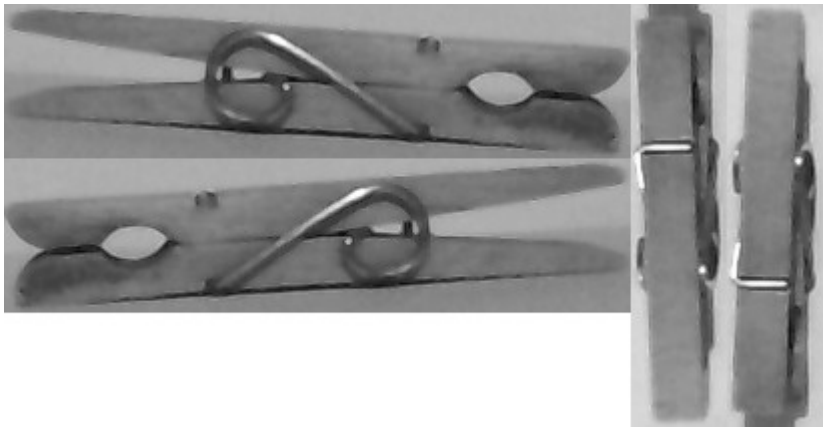
Kuva 24. Testikoulutuksen 3 tuloksia läheltä zoomattuna.

Saamme siis kaskaadimme huomaamaan oikeat objektit, ja jättämään samankaltaiset ei-halutut objektit huomiotta. Tämä kuitenkin toimii vain lähietäisyydeltä, ja johtuu havaintoikkunamme koosta, joka on kaikissa testeissämme ollut 20x20. Kauempaa katsottuna, tai pienellä havaintoikkunalla, Haar-tyyppiset piirteet näyttävät liian samoilta kameran kuvassa, ja tämä aiheuttaa väärät havainnot. Ikkunan koon kasvattamisen pitäisi auttaa siinä tapauksessa.

Monen alkuperäiskuvan käyttäminen voi myös auttaa vain halutun objektin havaitsemisessa, mutta siihen pääsemme tutustumaan myöhemmin. Kun käytämme montaa alkuperäiskuvaa, muodostamme jokaisesta halutun määrän esimerkkikuvia .vec-tiedostoihin, ja yhdistämme nämä tiedostot yhdeksi isoksi .vec-tiedostoksi käyttäen Naotoshi Seon *mergevec* -apuohjelmaa [15].

5.7 Testikoulutus 4 vaikeammalla 3d-objektilla

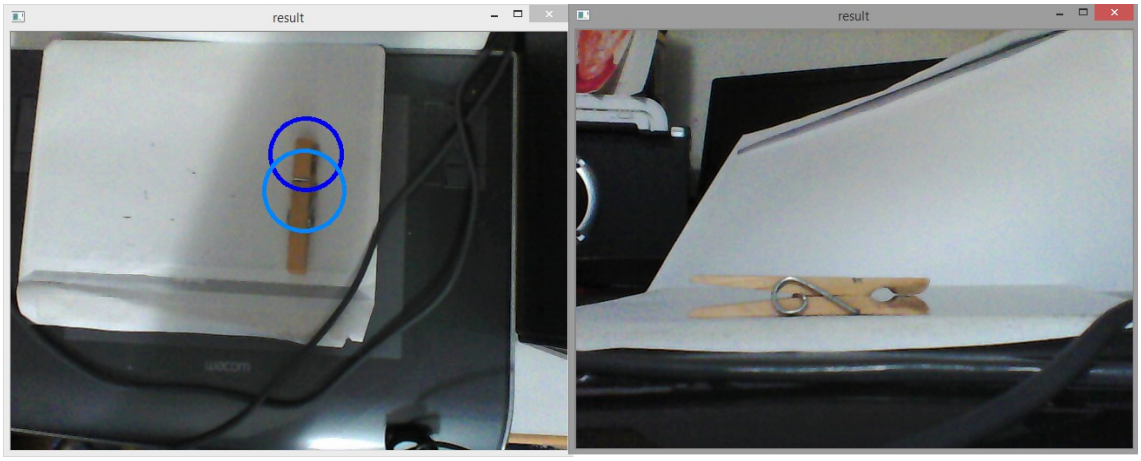
Tässä vaiheessa siirrymme 3d-objektin kouluttamiseen, joka on vaikeampaa kuin 2d-objektin koulutus. 3d-objekti pitäisi tunnistaa monilta puolilta, joten emme voi käyttää vain yhtä alkuperäiskuvaa, vaan joudumme käyttämään useita. Jos haluamme, että voimme tunnistaa jonkin objektiluokan, esimerkiksi kaikki banaanit, joutuisimme myös käyttämään monia eri banaaneja koulutuksessamme. Tässä ensimmäisessä 3d-objektin koulutuksessa käytämme kuitenkin vain yhtä objektia, josta otamme monta alkuperäiskuvaa. Kohteemme on tässä tapauksessa pyykkipoika, käyttäen kuvan 25 alkuperäiskuvia.



Kuva 25. 3d-objektin koulutuksen alkuperäiskuvat

Nyt käytössämme on monia alkuperäiskuvia, joten muodostamme niistä jokaisesta oman erillisen .vec-tiedoston *createsamples*-ohjelmalla, ja yhdistämme ne *mergevec*-apuohjelmalla. Käytämme testikoulutus 2 parametreja, vaihtaen vain määräksi 750 jokaiselle alkuperäiskuvalle, koska yhdistettyämme .vec-tiedostot tulee niitä olemaan 3000.

Koulutus kesti normaalin 7 tuntia ja saimme kuvan 26 kaltaisia tuloksia havainnoiksemme. Havaitseminen onnistuu siis vain, kun pyykkipoika on pystyasennossa tiettyyn suuntaan, ja sivusta havaitseminen ei onnistu laisinkaan. Tämä johtuu siitä, että Haar-tyyppiset piirteet, joita kaskaadi käyttää, ovat vain pystysuoria tässä tapauksessa. Lisäksi havaintoaika kasvaa kolminkertaiseksi, noin 120 ms - 200 ms aikaan. Voimme vetää tästä koulutuksesta siis kaksi selvää johtopäätöstä siitä, minkätyyppisiin objekteihin koulutus käytännössä soveltuu:



Kuva 26. Pyykkipojan havaitseminen kuvasta

1. Jos objekti on kolmiulotteinen, pitää objektin olla symmetrinen kaikilta puolilta, joilta haluamme sen havaita, koska luomme vain yhdenlaisia Haar-tyyppisiä piirteitä kaskaadiimme.
2. Jos haluamme havaita jotain tiettyä objektiluokkaa, paras tapa siihen on ottaa monta alkuperäiskuvaa objekteista, mutta vain yhdeltä puolelta objektia. Mieluiten siltä puolelta, jolta kaikki halutut objektit näyttävät samanlaisilta.

5.8 Testikoulutus 5 vaikeammalla 3d-objektilla

Aiemman koulutuksen tuloksien tietoja hyödyntäen luomme tässä koulutuksessa kaskaadin, jolla voimme havaita mukeja sivusta. Käytämme kuvan 27 alkuperäiskuvia koulutuksessamme.

Kuva 27. Testikoulutus 5 alkuperäiskuvat

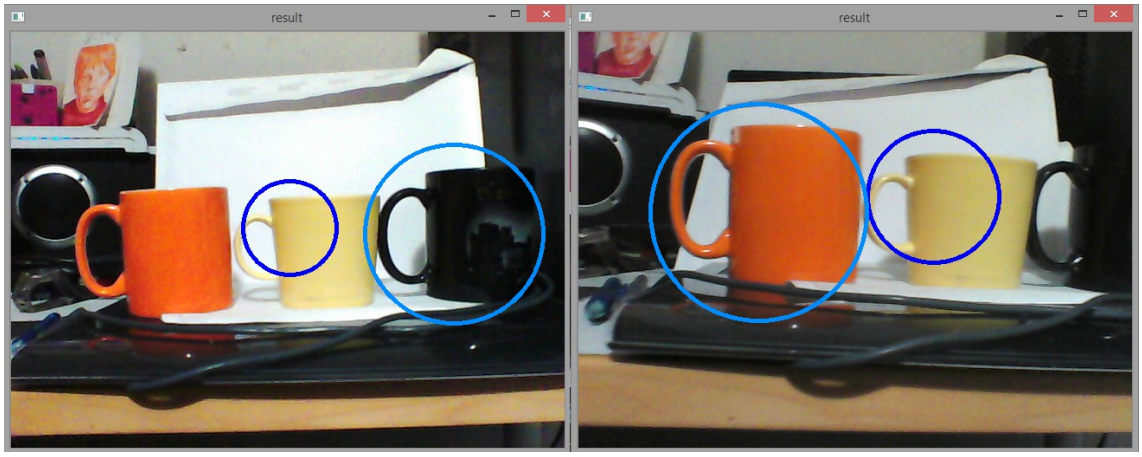
Luomme jokaisesta alkuperäiskuvasta 1000 esimerkkikuvaa, jotka yhdistämme yhdeksi .vec-tiedostoksi. Muina parametreinä *createsamples*-ohjelmassa käytämme testikou-

lutus 2 parametrejä. Käytämme myös koulutuksessa *createcascade*-ohjelmalla testikoulutus 2 parametrejä, muuttaen negatiivisten määrän 3000:ksi ja positiivisten määräksi 2800. Koska käytössämme on yhteensä 3000 esimerkkiä, tämä positiivisten määrä pitäisi olla sopiva.

```
opencv_traincascade -data classifier5 -vec mugs.vec -bg
test2negatives/test2negatives.dat -numStages 20 -minHitRate
0.999 -maxFalseAlarmRate 0.5 -numPos 2800 -numNeg 3000 -w 20 -h
20 -mode ALL -precalcValBufSize 1024 -precalcIdxBufSize 1024
```

Listaus 7. Traincascaden parametrit testikoulutuksessa 5.

Koulutuksen kesto oli tässä n. 8 h, ja tulokseksi saimme kaskaadin, joka pystyy huomaamaan mukimme, kunhan ne ovat samassa asennossa kuvan 28 mukaisesti.



Kuva 28. Mukikoulutuksen tuloksia.

Koska koulutuksesta saadut Haar-tyyppiset piirteet ovat suunnilleen samoja kaikilla korvallisilla mukeilla, voimme havaita muitakin mukeja koulutuksesta saadulla kaskaadilla kuvan 29 mukaisesti.

Kuva 29. Mukikoulutuksen tuloksia

Pystymme siis Haar-koulutuksella saamaan samantyyppisiä piirteitä sisältävät objektit, siis yhden tietyn "objektiluokan", tunnistettavaksi yhdellä koulutuksella. Tein vielä ylimääräisen koulutuksen parantamaan tätä toimintaa, sillä nyt havaintonopeus on 125 ms - 170 ms ja ohjelma antaa liikaa vääriä havaintoja. Tämä koulutus nopeutti havaintoa yli puolella, antaen n. 55 ms - 60 ms havaintoajan ja vähensi väärien havaintojen määrää, mutta vähensi myös oikeiden havaintojen määrää. Tämä johtui alhaisemmasta värien maksimivahvuuden erosta, joka ensimmäisessä mukien koulutuksessa oli 60, ja jälkimmäisessä 40. Kuten värien maksimivahvuuksia, myös kääntökulmia pienennettiin jälkimmäisessä koulutuksessa. Jälkimmäisessä koulutuksessa kesti n. 21 h.

6 Keskustelua

6.1 Esimerkkikoulutusten toiminta.

Koulutimme työssä kahdenlaisia kaskaadeja: Yksinkertaisia 2d-objektin Haar-tyyppisiä piirteitä sisältäviä, sekä monimutkaisempia 3d-objektin Haar-tyyppisiä piirteitä sisältäviä kaskaadeja. 2d-objektin, eli tämän työn tapauksessa paperille piirretyn hymiön, koulutus sujui varsin moitteettomasti, ja sen havainnointi kameran kuvasta oli jo testikoulutuksen 2 jälkeen erittäin helppoa ja nopeaa. Ongelmia tässä koulutuksessa muodostui silloin, jos haluamme että samantyyppisiä piirteitä sisältävät objektit jätettäisiin havaitsematta. Koska Haar-koulutus opettaa vain yleisesti positiivisissa kuvissa vastaan tulevia piirteitä, ja vain havaintoikkunan koon mukaisesti, on varsin vaikeaa jättää samankaltaiset objektit havaitsemisen ulkopuolelle. Tämä on mahdollista, mutta vaatii lisätestausta ja koulutusta, jotta saamme halutun tasapainon oikean objektin havaitsemiselle ja samankaltaisten väärien objektien havaitsematta jättämiselle.

3d-objektien havaitseminen on jo astetta vaikeampaa, joskin seuraa samoja askelia kuin 2d-objektin havaitseminen. Koska käytämme vain opetettuja piirteitä, emme käytännössä voi havaita objektia monesta eri kulmasta käyttäen Haar-koulutusta. Jos koitamme opettaa kaskaadin havaitsemaan epäsymmetristä objektia monesta eri kulmasta, tulevat kaskaadille opetetut Haar-piirteet olemaan erittäin epäoptimaaliset. Haar-koulutus toimii toisaalta erittäin hyvin, jos haluamme opettaa jonkin tietyn objektien luokan kaskaadillemme, koska nämä objektit sisältävät paljon samoja piirteitä. Mukikoulutus toimi erittäin hyvin, ja pystyi havaitsemaan muitakin samanlaisia mukeja kuin mitä

olimme kaskaadin alkuperäiskuvina käyttäneet. Tämä toimi kuitenkin vain tapauksessa, jossa mukit olivat samassa asennossa kuin alkuperäiskuvissa.

6.2 Haar-koulutuksen haasteita

Suurimpia haasteita Haar-koulutuksessa ovat koulutukseen kuluva aika, sekä piirteiden epäsymmetrisyys. Työssä käytettyihin kuuteen testikoulutukseen kului yhteensä aikaa n. 70 tuntia, joten jo pelkkään testaukseen kuluu hirvittävästi aikaa. Lyhin yksittäinen koulutus työssä vei 3 tuntia, ja pisin vei 21 tuntia. Tämä aiheuttaa monia ongelmia tämän koulutusmetodin käytössä, varsinkin koska koulutuksen aikana kouluttava kone on miltei täysin käyttökelvoton mihinkään muuhun. *Opencv_traincascade*-ohjelma on kuitenkin helpottanut tätä ongelmaa jonkin verran vanhemmasta, *haarcascade*-ohjelmasta antamalla sen käyttää monisäikeisyyttä. Vanhemmalla ohjelmalla kouluttaminen vie paljon enemmän aikaa.

Haar-tyyppisten piirteiden epäsymmetrisyys on toinen suuri haaste tämän koulutusmetodin käytössä. Jos koulutettu objekti on epäsymmetrinen, voimme usein havaita sen vain kun objekti on tietyssä kulmassa, tai osoittaa tiettyyn suuntaan. Tätä voidaan helpottaa määrämällä kiertokulmia *opencv_createsamples*-ohjelmassa, mutta se auttaa vain tiettyyn pisteeseen saakka.

Tämä koulutusmetodi soveltuukin pääasiassa sellaisten objektien havaitsemiseen, jotka tulevat havaitsevan kameran eteen edeltäkäsin määritellyssä kulmassa, tai ovat joka kulmasta täysin symmetrisiä. Kuten 2d-objektin koulutuksessa saimme huomata, pystyimme helpon objektin havaitsemaan vaikka se annettiin kameralle 90-asteen kulmassa, joten tässä tilanteessa pelkät *opencv_createsamples*-ohjelman kiertokulmien säädöt auttoivat havainnointia.

6.3 Muita OpenCV:n tukemia koulutusmetodeja

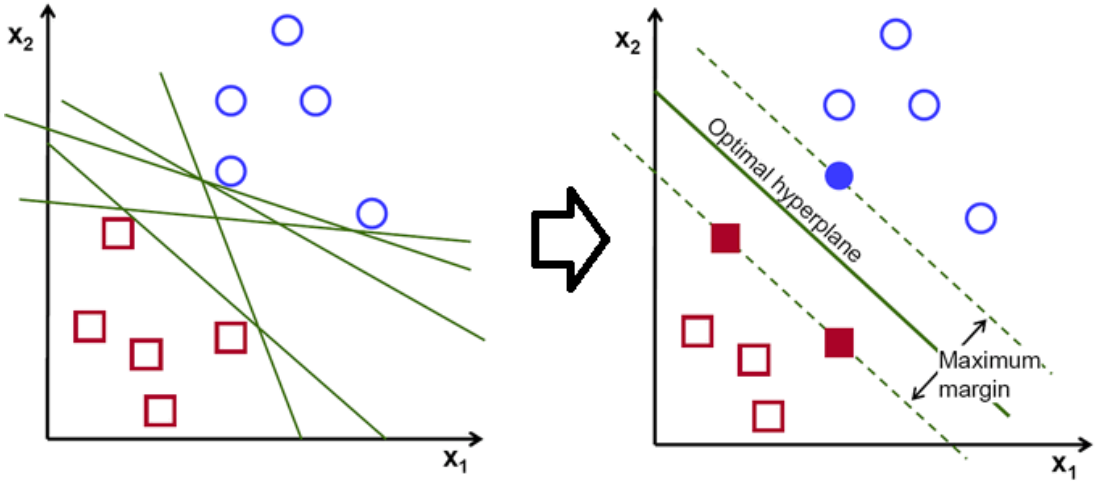
Haar-koulutuksen lisäksi OpenCV tukee muitakin koneoppimisen (Machine Learning) algoritmejä. Mikään mainituista algoritmeista ei ole "paras", vaan niiden käyttökelpoisuus riippuu täysin käyttäjän tilanteesta. Jos käyttäjällä on paljon aikaa koulutukseen, ja tarvitsee korkean tarkkuuden, boostaus ja satunnaiset puut ovat erittäin hyviä. Jos taas luokittaja pitää kouluttaa nopeasti, nearest neighbors, normal Bayes tai päätöspuut ovat hyviä vaihtoehtoja. [1, s. 465.]

Support Vector Machine, SVM

Tämä algoritmi on vahva syrjivä luokittaja, joka on muodollisesti määritelty erottavalla hypertasolla. Toisin sanoen jos sille antaa kaksikategorisen leimatun koulutusdatan, algoritmi antaa tuloksena optimaalisen hypertason, joka kategorisoi uusia esimerkkejä [26].

Määritämme kuvat siis kahteen eri kategoriaan, esimerkiksi negatiiviseen ja positiiviseen, ja SVM laskee mahdollisimman laajan hypertason näiden kategorioiden välille. Uudet kuvat jotka saadaan esimerkiksi kameralta pyritään ennustaen jakamaan piirteidensä mukaisesti jompaankumpaan kategoriaan. SVM:ssä on omat tekniset eronsa Haar-koulutukseen, mutta periaate on sama. OpenCV sisältää funktion *CvSVM::train,* jolla voimme kouluttaa halutunlaisen SVM-algoritmin. *CvSVM{}*-luokka sisältää kaikki tähän algoritmiin liittyvät funktiot. [1, s. 517; 1, s. 463.]

SVM on varsin hyvä jos datakokoelma on rajoitettu, mutta muussa tapauksessa boostaus tai satunnaiset puut ovat tehokkaampia.



Kuva 30. SVM:n laskema optimaalinen hypertaso [26]

OpenCV tukee muitakin koneoppimisen algoritmeja. Niitä ovat *Expectation Maximization, K-Nearest Neighbors* sekä *Multilayer Perceptron.*

Expectation Maximization

Haar-koulutuksen tapaan klusterointitekniikka. Käsittää monta iteraatiota, joissa otetaan kaikkein todennäköisin arvaus määritellyn mallin mukaisesti, ja sitten säätäen mallia maksimoidakseen oikeassa olemisen. OpenCV:ssä tämä algoritmi sisältyy *CvEM{}*-

luokkaan, ja yksinkertaisesti asettaa Gaussiaanisen seoksen datan sekaan. [1, s. 516; 1, s. 463.]

K-Nearest Neighbors

Yksi yksinkertaisimmista luokittelutekniikoista, joka vain sisältää kaikki koulutuksen datapisteet. Kun uusi piste halutaan luokittaa, etsitään sille *K*(kokonaisluku) lähintä pistettä ja sitten leimataan uusi piste sen mukaan, mikä kokoelmista sisältää suurimman osan sen *K* naapureita. OpenCV:ssä tätä algoritmia voidaan käyttää *CvKNearest{}* -luokalla. [1, s. 516; 1, s. 463.]

Multilayer Perceptron

Tämä on neuroverkko, joka on yksi parhaiten toimivista luokittajista, varsinkin tekstin tunnistuksessa. Voi olla koulutuksessa hidas, mutta on testitilassa nopea. OpenCV:ssä tätä algoritmia voidaan käyttää *CvANN_MLP{}* -luokan kautta. [1, s. 517; 1, s. 463.]

Muita OpenCV:n tukemia koneoppimisen algoritmeja ovat *Mahalanobis*, *K-means*, *Normal/Naive Bayes classifier*, *Decision trees*, *Boosting* sekä *Random trees*. [1, s. 462 - 463.]

6.4 Koulutuksen jatkokehitys

Viimeisen koulutuksen havaintokykyä voimme parantaa kasvattamalla värien maksimivoimakkuuseroa, testaamalla paremmilla taustoilla ja ottamalla mukaan useampia alkuperäiskuvia. Kun olemme saaneet parhaan tasapainon väärien havaintojen ja oikeiden objektiluokkaan kuuluvien objektien, eli meidän tapauksessamme mukien, voimme teoriassa soveltaa tämän saman koulutuksen parametreja ja kaikkien kuvien määrää mihin tahansa muuhun objektiin. Mukihavaitsimen lisäksi voisimme siis tehdä vaikkapa banaanihavaitsimen, joskin värien erotus ei vaikuta niihin yhtä paljon kuin mukien havaitsemisessa.

Voisimme myös kehittää ohjelman, joka pystyisi käyttämään montaa kaskaaditiedostoa samanaikaisesti, ja antaa sille monia kaskaadeja, jotka kuvaavat haluttua objektia eri puolilta. Täten pystyisimme tunnistamaan epäsymmetrisiä objekteja riippumatta siitä, miten päin ne ovat kameran edessä.

7 Lopputulokset

Tämän työn päätarkoituksena oli selvittää Haar-koulutuksen toimintaa teorian ja käytännön kokeiden avulla, ja saada aikaan kaskaadeja, joiden avulla voimme testata tämän koulutusmetodin toiminnallisuutta erilaisten objektien tunnistuksessa. Halusin selvittää itselleni myös tietokonenäön toimintaa perustasolla, mutta päätarkoitus oli kuitenkin Haar-koulutuksessa, sen toiminnassa ja testauksessa.

Vaikka olin tutustunut aikaisemmin tähän koulutusmetodiin, oli sen tarkemmin tutkiminen haastava työ. Varsinkin teoria oli haastava asia, ja tuntuu, että sitä voisi tutkia vielä paljonkin enemmän. Käytännön testeissä haastavia asioita riitti melkein sitäkin enemmän, sillä pienetkin virheet koulutuksien parametreissä tarkoittivat tuntien koulutusajan hukkaan heittämistä. Koska tarkimmat tulokset saadaan vain suurilla määrillä alkuperäis- ja esimerkkikuvia, ja näiden lisääminen kasvattaa merkittävästi koulutusaikaa, oli tämä vaikea asia tasapainottaa.

Koulutuksen voi myös suorittaa käyttäen LBP:tä Haarin sijasta *createcascade*-ohjelmassa, ja tämä nopeuttaisi niin koulutusta kuin havainnointiakin. LBP käyttää kuitenkin vain kokonaislukuarvoja piirteiden laskussa, joten tarkkuus voisi olla alhaisempi. Hyvällä koulutusdatalla ja parametreillä voimme päästä LBP:llä lähelle Haarin tarjoamaa tarkkuutta.

Käyttämäni metodit eivät varmasti olleet parhaita työhön, ja monta kertaa eteen tulikin ongelmia, jotka olisin voinut välttää tekemällä jonkin asian eri tavalla. Yksi tapa testata nopeasti erilaisia koulutuksia eri parametreilla olisi käyttää jotain pilvipalvelua ja virtuaalikoneita tämän tekemiseen; Tällä tavalla voisimme ajaa testejä rinnakkain paljon nopeammin kuin vain yhtä tai kahta konetta käyttäen.

Olen kyllä yllättynyt, kuinka helposti lopulta sain koulutukset toimimaan, vaikka niissä aikaa kestikin. Alunperin odotin koulutusten pelkkien valmistelujen vievän kaksinkertaisen ajan, koulutukseen itseensä kuluvasta ajasta puhumattakaan.

Lähteet

1. Bradski G., Kaehler A. 2008. Learning OpenCV. O'Reilly Media Inc.
2. Azad P., Gockel T., Dillmann R. 2008. Computer Vision Principles and Practice. Elektor International Media.
3. Kaksonen I. 2013. Using Computer Vision to Detect and Calculate the Number of Objects from the Source Image. Pro gradu.
4. Itseez. About | OpenCV [verkkodokumentti]. URL: http://opencv.org/about.html. Päivitetty 21. elokuuta 2014. Luettu 24. lokakuuta 2014.
5. Itseez. OpenCV | OpenCV [verkkodokumentti]. URL: http://opencv.org/. Päivitetty 21. elokuuta 2014. Luettu 24. lokakuuta 2014.
6. Itseez. OpenCV Documentation | OpenCV [verkkodokumentti].URL: http://docs.opencv.org/doc/user_guide/user_guide.html. Päivitetty 21. huhtikuuta 2014. Luettu 24. lokakuuta 2014.
7. Itseez. OpenCV Thresholding | OpenCV [verkkodokumentti]. URL: http://docs.opencv.org/doc/tutorials/imgproc/threshold/threshold.html. Päivitetty 21. huhtikuuta 2014. Luettu 24. lokakuuta 2014.
8. Itseez. OpenCV Thresholding Tutorial | OpenCV [verkkodokumentti]. URL: http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html. Päivitetty 7. lokakuuta 2014. Luettu 8. lokakuuta 2014.
9. Itseez. OpenCV Morphology | OpenCV [verkkodokumentti]. URL: http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html. Päivitetty 21. huhtikuuta 2014. Luettu 28. lokakuuta 2014.
10. Codersource. Binary Image | Binary Image [verkkodokumentti]. URL: http://web.archive.org/web/20080610170124/http://www.codersource.net/csharp_color_image_to_binary.aspx.Päivitetty 18. toukokuuta 2005.Luettu 28. lokakuuta 2014.
11. Keyetech. IVT | Integrating Vision Toolkit [verkkodokumentti]. URL: http://ivt.sourceforge.net/. Päivitetty 8. toukokuuta 2014. Luettu 28. lokakuuta 2014.

12. Baggio D.L., Emami S., Escrivá D.M., Ievgen K., Mahmood N., Shilkrot R. 2012. Mastering OpenCV with Practical Computer Vision Projects. Packt Publishing.

13. Lienhart R., Kuranov A., Pisarevsky V. 2002. Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection.

14. Viola P., Jones M. 2001. Rapid Object Detection using a Boosted Cascade of Simple Features.

15. Seo N. Haartraining | OpenCV Haartraining [verkkodokumentti]. URL: http://note.sonots.com/SciSoftware/haartraining.html. Päivitetty 11. huhtikuuta 2012. Luettu 1. marraskuuta 2014.

16. Itseez. Face Detection using Haar Cascades | OpenCV Haartraining [verkkodokumentti]. URL: http://docs.opencv.org/master/doc/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html. Päivitetty 7. marraskuuta 2014. Luettu 8. marraskuuta 2014.

17. Coding Robin. Haartraining | OpenCV Haartraining [verkkodokumentti]. URL: http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html. Päivitetty 8. syyskuuta 2014. Luettu 1. marraskuuta 2014.

18. OpenCV Contours | OpenCV [verkkodokumentti]. URL: http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html. Päivitetty 21. Huhtikuuta 2014. Luettu 2. marraskuuta 2014.

19. Teh C.H., Chin R.T. 1989. On the Detection of Dominant Points on Digital Curve. Pp 859-872.

20. HIPR2. Pixel Connectivity | Image Processing Learning Resources [verkkodokumentti]. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/connect.htm. Päivitetty 1. maaliskuuta 2004. Luettu 4. marraskuuta 2014.

21. Papageorgiou, Oren, Poggio. 1998. A general framework for object detection. International Conference on Computer Vision.

22. Itseez. Cascade Classification | OpenCV [verkkodokumentti]. URL: http://docs.opencv.org/master/modules/objdetect/doc/cascade_classification.html?highlight=haar. Päivitetty 7. marraskuuta 2014. Luettu 8. marraskuuta 2014.

23. Itseez. Boosting | OpenCV [verkkodokumentti]. URL: http://docs.opencv.org/modules/ml/doc/boosting.html. Päivitetty 21. Huhtikuuta 2014. Luettu 4. marraskuuta 2014.

24. Itseez. Training Cascade | OpenCV [verkkodokumentti]. URL: http://docs.opencv.org/doc/user_guide/ug_traincascade.html. Päivitetty 21. Huhtikuuta 2014. Luettu 5. marraskuuta 2014.

25. Kuranov A., Lienhart R., and Pisarevsky V. 2002. An Empirical Analysis of Boosting Algorithms for Rapid Objects With an Extended Set of Haar-like Features. Intel Technical Report MRL-TR-July02-01.

26. Itseez. Support Vector Machine | OpenCV [verkkodokumentti]. URL: http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html#introductiontosvms. Päivitetty 21. Huhtikuuta 2014. Luettu 5. marraskuuta 2014.

Käytettyjä negatiivisia kuvia